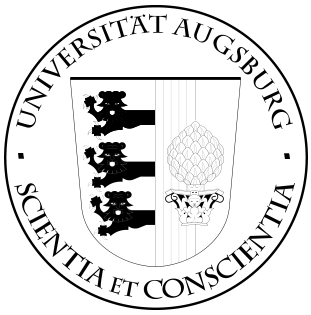
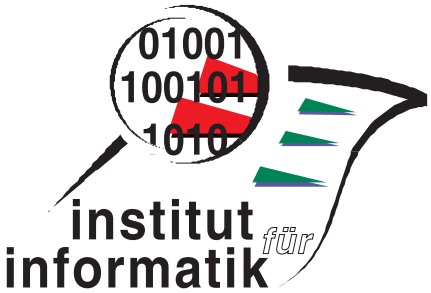
Universität Augsburg



Improved Decomposition of Signal Transition Graphs

Walter Vogler and Ben Kangsah

Report 2004–8 April 2004



Institut für Informatik
D-86135 Augsburg

Improved Decomposition of Signal Transition Graphs *

W. Vogler
Institut für Informatik
Universität Augsburg
vogler@informatik.uni-augsburg.de

B. Kangsah
FB Elektro- und Informationstechnik
Technische Universität Kaiserlautern
kangsah@rhrk.uni-kl.de

Abstract

Signal Transition Graphs (STGs) are a version of Petri nets for the specification of asynchronous circuit behaviour. It has been suggested to decompose such a specification as a first step; this leads to a modular implementation, which can support circuit synthesis by possibly avoiding state explosion or allowing the use of library elements.

In a previous paper, the original method was extended and shown to be much more generally applicable than known before. But further extensions are necessary, and some are presented in this paper, e.g.: to avoid dynamic auto-conflicts, the previous paper insisted on avoiding structural auto-conflicts, which is too restrictive; we show how to work with the latter type of auto-conflicts. This and another simple extension makes it necessary to restructure presentation and correctness proof of the decomposition algorithm.

1 Introduction

Decomposition of Signal Transition Graphs (STGs) has been suggested as a method to alleviate the problem of state space explosion. As a significant improvement compared to previous efforts, it was shown in [VW02] with a comparatively simple correctness proof that decomposition can be applied quite generally. Nevertheless, the study of benchmark examples has revealed that further improvements of the method are needed. In the present paper, we give several improvements, also restructuring the correctness proof in a way that should make the incorporation of further improvements easier. Since this paper builds so much on [VW02], there are numerous quotations from [VW02]; we even have to repeat some proofs that we expand to give new practically relevant results.

STGs, see e.g. [Wen77, RY85, Chu86], are a version of Petri nets for the specification of asynchronous circuit behaviour; they are supported by the tools petrify (e.g. [CKK+97]) and CASCADE [BEW00], which in many cases can synthesize a circuit

*This work was partially supported by the DFG-project 'STG-Dekomposition' Vo615/7-1 / Wo814/1-1.

from an STG. The transitions are labelled with the rising and falling edges of input and output signals; the latter are thought to be controlled by the circuit, the former by its environment. If the occurrence of an input signal in some state is not specified, this formulates the assumption on the environment not to produce this signal; this is in contrast e.g. to I/O-automata [Lyn96].

To study the principle ideas of decomposition, STGs were only labelled with signals instead of signal edges in [VW02]. Here, we consider signal edges; in practical STGs, rising and falling edges of a signal have to alternate, i.e. the STGs have to be *consistent*. As one improvement we show in the present paper that decomposition of a consistent STG results in consistent components.

Being Petri nets, STGs allow a causality-based specification style, and they give a compact representation of the desired behaviour since they represent concurrency explicitly. As a first step in the synthesis of a circuit corresponding to a given STG N, one usually constructs the reachability graph, where one might encounter the state explosion problem; i.e. the number r of reachable states (markings) might be too large to be handled. To avoid this, one can try to decompose the STG into components C_i; their reachability graphs taken together can be much smaller than r since r might be the product of their sizes. Even if this is not achieved, it might already be interesting enough if each component has a smaller reachability graph: the reachability graph of N might be too large for the available memory space; even if memory is no limiting factor, further steps of the circuit synthesis might easily take quadratic time in the number of states. Decomposition can also be useful independently of size considerations: there are examples where N cannot be handled by a specific synthesis method, while the C_i can; also, one may be able to split off a library element, and this is valuable in particular for arbiters, which are complicated to synthesize; see [VW02] for an example.

Thus, instead of synthesizing one large circuit from N, we decompose N into components C_i, synthesize a circuit from each C_i (e.g. using tools or library look-ups) and compose these circuits into one system of communicating circuits.

[Chu87a, Chu87b, KKT93] suggest decomposition methods for STGs, but these approaches can only deal with very restricted net classes. [VW02] deals with STGs of arbitrary graph-theoretic structure, but with some limitations on the labelling. While there is not even a correctness definition in [KKT93], [Chu87a] proves that the parallel composition of the components generated by decomposition of N is language equivalent to N. In [VW02], it is argued that instead of language equivalence a new bisimulation-type relation is more adequate, and this correctness definition is also used here.

The method in [Chu87a] constructs for each output signal s a component C_i that generates this signal; C_i has as inputs all signals that – according to the net structure – may directly cause s. The component is obtained from the STG N by contracting all transitions belonging to the signals that are neither input nor output signals for this component; these transitions are internal in intermediate stages of the decomposition. So-called *secure* transition contraction of internal transitions is also the main operation in [VW02], and components may also generate several outputs as in [KKT93]. An additional operation is the deletion of redundant places (see e.g. [Ber87] for an early reference), which is already essential for the decomposition of the very simple marked

graphs. For a more detailed discussion of the literature, see [VW02].

Our improvements of the decomposition method from [VW02] are motivated by our study of examples from a collection of *benchmark examples* that circulate in the STG-community. For example, the initial STG N was required to be deterministic in [VW02], but some benchmark examples have internal (or dummy) transitions; it is a simple observation that we can try to remove them with secure transition contractions, which works in many cases. Further, it was required that N has no structural conflicts between input and output signals; this ensures that there are no dynamic conflicts between inputs and outputs, which in principle make it impossible to turn N into a reliable (i.e. hazard-free) digital circuit. The whole correctness proof in [VW02] makes this assumption, and ensures that there are no such structural conflicts in the constructed components. Motivated by an example, we untangle the proof to demonstrate: correctness of decomposition does not depend on this assumption; the constructed components only have structural or dynamic conflicts between input and output signals, if N has such conflicts between the same signals.

The notion of an admissible operation was introduced in [VW02] in order to structure the correctness proof, also with an eye on possible extensions of the method. From our study of examples, we found that we should add another simple operation besides secure transition contraction and deletion of redundant places: the deletion of internal transitions that are connected to places only by loops. This operation is quite trivial, but we could not insert it directly into the correctness proof of [VW02]. Therefore, we had to restructure this proof including a change to the notion of admissible operation, and we present decomposition in this paper in a slightly different way. This should also make further possible additions to the list of operations easier.

The rewritten proof also supports our main contribution concerning auto-conflicts, i.e. conflicts between transitions labelled with the same signal edge. If such a conflict is dynamic, then the STG is not deterministic and cannot be implemented directly as a circuit; a simple condition that ensures the absence of dynamic auto-conflicts without generating the reachability graph is to require the absence of structural auto-conflicts. Therefore, it is required in [VW02] that the initial STG is free of structural auto-conflicts; if a transition contraction introduces such a conflict, some form of backtracking is applied. There are examples where there are structural auto-conflicts initially which are not dynamic; decomposition as in [VW02] cannot handle such examples.

We show that decomposition can be applied in such cases; if the contractions do not introduce new structural auto-conflicts (something that can be checked locally), the resulting components will not have a dynamic auto-conflict. Similarly, we can carry on without backtracking if a structural auto-conflict is not a dynamic one; this has the potential for better results, since backtracking leads to components with more signals and, thus, larger reachability graphs.

For the time being, we assume that the user has to ensure that a structural auto-conflict is not a dynamic one. Such user intervention is of course error-prone, and in the presence of dynamic auto-conflicts the correctness proof fails, so there is the danger that the result of the decomposition will exhibit incorrect behaviour. A nice final touch to our presentation is the following result: if due to an error, a dynamic auto-conflict is present in an intermediate stage of some component, then this is

essentially preserved and therefore also present in the final component. Since for circuit synthesis the reachability graph of this component will be built, the conflict will be discovered and the utilization of a faulty component can be avoided.

After presenting basic definitions of STGs in Section 2, we have a closer look at contractions in Section 3 refining the results of [VW02]. In Section 4, we give the new description of our method in detail. The new features of the method have been integrated in our tool DESI; we present some first results of our improvements in Section 5. Topics for future research are discussed in the conclusion in Section 6.

We thank Jordi Cortadella, Mark Schäfer and Ralf Wollowski for very helpful discussions, and Josep Carmona and Ivan Blunno for providing us with the benchmark examples.

2 Basic Notions of Signal Transition Graphs

In this section, we introduce the kind of Petri nets we study in this paper, some standard behaviour notions, and the operation of parallel composition. For general information on ordinary Petri nets, the reader is referred to e.g. [Pet81, Rei85]. A *Signal Transition Graph* or *STG* is a net that models the desired behaviour of an asynchronous circuit. Its transitions are labelled with edges of signals from some alphabet Σ or with the empty word λ, and we distinguish between input and output signals. A transition labelled with λ represents an internal, unobservable signal, which could be an internal signal between components of a circuit. In this paper, we use λ-labelled transitions only in the initial STG, where we call them *dummy transitions* which have to be removed in a first phase, or in intermediate phases of our algorithm, where we call them *divining* transitions.

Thus, an *STG* $N = (P, T, W, l, M_N, In, Out)$ is a labelled net consisting of finite disjoint sets P of *places* and T of *transitions*, the *arc weight* $W : P \times T \cup T \times P \to \mathbb{N}_0$, the *labelling* $l : T \to In\{+,-\} \cup Out\{+,-\} \cup \{\lambda\}$, the *initial marking* $M_N : P \to \mathbb{N}_0$ and the disjoint sets $In \subseteq \Sigma$ and $Out \subseteq \Sigma$ of *input* and *output signals*; $\mathbb{N}_0$ denotes the natural numbers including 0. Usually, an STG is required to be *consistent* (i.e. signal edges are required to alternate) as defined below.

We usually use a, b, c for input and x, y, z for output signals; $In\{+,-\} = \{a+, a- \mid a \in In\}$ is the set of input signal edges, where $a+$ is the rising and $a-$ the falling edge of signal a – and the meaning of $Out\{+,-\}$ or $\Sigma\{+,-\}$ is analogous. For signal s, we write $s\pm$ for any one of its edges if the direction does not matter; writing $s\pm$ several times in some context refers to the same edge of s. If $l(t) = s\pm$, then s is the *signal of* t and t is *observable*; if $s \in In$ ($s \in Out$ resp.), then t is an input (an output resp.) transition, drawn as a black (a white resp.) box; if $l(t) = \lambda$, then t is an *internal* transition, drawn as a line or a box with two lines in it. When we introduce an STG N or N_1 etc., then we assume that implicitly this introduces its components P, T, W, ... or P_1, T_1, ... etc.

We say that there is an *arc* from $x \in P \cup T$ to $y \in P \cup T$ if $W(x, y) > 0$. For each $x \in P \cup T$, the *preset* of x is ${}^\bullet x = \{y \mid W(y, x) > 0\}$ and the *postset* of x is $x^\bullet = \{y \mid W(x, y) > 0\}$. If $x \in {}^\bullet y \cap y^\bullet$, then x and y form a *loop*. A *marking* is a function $P \to \mathbb{N}_0$ giving for each place a number of *tokens*. We now define the basic firing rule.

- A transition t is *enabled* under a marking M, denoted by $M[t\rangle$, if $W(.,t) \leq M$. If $M[t\rangle$ and $M' = M + W(t,.) - W(.,t)$, then we denote this by $M[t\rangle M'$ and say that t can *occur* or *fire* under M yielding the follower marking M'.

- This definition of enabling and occurrence can be extended to sequences as usual: a finite sequence $w \in T^*$ of transitions is *enabled* under a marking M, denoted by $M[w\rangle$, and yields the follower marking M' when *occurring*, denoted by $M[w\rangle M'$, if $w = \lambda$ and $M = M'$ or $w = w't$, $M[w'\rangle M''$ and $M''[t\rangle M'$ for some marking M'' and transition t. If w is enabled under the initial marking, then it is called a *firing sequence*.

- We extend the labelling to sequences of transitions as usual, i.e. $l(t_1 \dots t_n) = l(t_1) \dots l(t_n)$; note that internal signals are automatically deleted in this *image* of a sequence. With this, we lift the enabledness and firing definitions to the level of signal edges: a sequence v of signal edges from $\Sigma\{+,-\}$ is *enabled* under a marking M, denoted by $M[v\rangle\rangle$, if there is some transition sequence w with $M[w\rangle$ and $l(w) = v$; $M[v\rangle\rangle M'$ is defined analogously. If $M = M_N$, then v is called a *trace*. The *language* $L(N)$ is the set of all traces. We call two STGs *language equivalent* if they have the same traces.

  An STG is *consistent* (which is usually required) if, for all signals s, in every trace of the STG the edges $s+$ and $s-$ alternate and there are no two traces where $s+$ comes first in the one and $s-$ in the other.

- A marking M is called *reachable* if $M_N[w\rangle M$ for some $w \in T^*$. The *reachability graph* of N consists of the reachable markings as vertices (with M_N as a designated initial vertex) and with an edge from M to M' whenever $M[t\rangle M'$ for some transition t; such an edge is labelled t or $l(t)$, depending on the context. The STG is *k-bounded* if $M(p) \leq k$ for all places p and reachable markings M; it is *safe* if it is 1-bounded and *bounded* if it is k-bounded for some k.

Often, STGs are assumed to be safe and to have only arcs with weight 1. In the first place, we are interested in such STGs; but we also deal with bounded STGs with larger arc weights, in particular since they can turn up in our decomposition algorithm. Note that there is no additional problem to synthesize a circuit from such an STG if the reachability graph is used as an intermediate construction [VYCLdM94, Wol97].

W.r.t. consistency, we have the following obvious result:

Lemma 2.1 *Let N and $\overline{N}$ be STGs with N being consistent. If $L(\overline{N}) \subseteq L(N)$, then $\overline{N}$ is consistent, too.*

The idea of input and output signals is that only the latter are under the control of the circuit modelled by an STG. The STG requires that certain outputs (or, more precisely, output signal edges) are produced provided certain inputs have occurred, namely those outputs that are enabled under the marking reached by the signal occurrences so far. At the same time, the STG describes assumptions about the environment that controls the input signals: if some input signal is not enabled, the environment is supposed not to produce this input at this stage; if it does, the

specified system may show arbitrary behaviour, and it might even malfunction. Inputs and outputs will become really important in Section 4.

In this paper, we mainly deal with specifications that completely specify the desired behaviour in the sense of determinism: an STG is *deterministic* if it does not have internal transitions and if for each of its reachable markings and each signal edge $s\pm$, there is at most one $s\pm$-labelled transition enabled under the marking. It is useful to distinguish two forms how determinism can be violated.

- Two different transitions t_1 and t_2 are *enabled concurrently* under a marking M if $W(.,t_1)+W(.,t_2) \leq M$, i.e. if there are enough tokens for both transitions together. If both transitions are labelled with the same signal edge $s\pm \in \Sigma\{+,-\}$, then s is *enabled auto-concurrently* under M. (Note that this cannot happen in a consistent STG; there, t_1 and t_2 cannot even have the labels $s+$ and $s-$. With the former observation, some proofs of the present paper could be simplified; we try to make little use of consistency in our proofs in order to allow application of our results outside the area of circuit design.) An STG is *without auto-concurrency*, if no signal is enabled auto-concurrently under any reachable marking.

- Two different transitions t_1 and t_2 are *in conflict* under a marking M if they are not enabled concurrently under M, but $M[t_1\rangle$ and $M[t_2\rangle$. If both transitions are labelled with the same signal edge $s\pm \in \Sigma\{+,-\}$, then s is *in auto-conflict* under M and the STG has a *(dynamic) auto-conflict* if M is reachable. (Note that, for a consistent STG, t_1 and t_2 cannot have the labels $s+$ and $s-$.) If the signal of one of the transitions is an input signal, while the signal of the other is an output signal, then there is an *input/output-conflict* under M and the STG has a *(dynamic) input/output-conflict* if M is reachable.

- Two different transitions t_1 and t_2 – and also their signals – are *in structural conflict* if ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$. If both transitions are labelled with the same signal edge $s\pm \in \Sigma\{+,-\}$, then s is *in structural auto-conflict* and the STG *has* such a conflict. If t_1 is an input (or a λ-labelled) and t_2 an output transition, then they form a *structural input/output conflict* (or a *structural λ/output conflict*) and the STG *has* such a conflict.

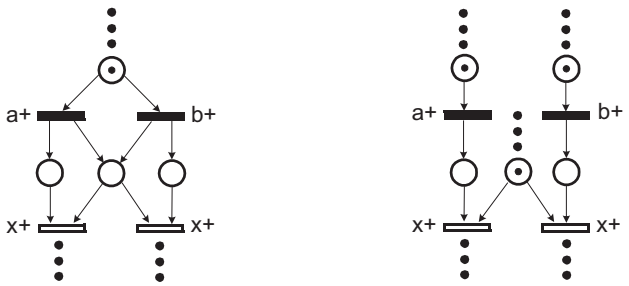


Figure 1

Figure 1 shows on the left an STG with a structural but without a dynamic auto-conflict. The STG on the right has a dynamic (and thus also a structural) auto-conflict; without the marked place in the middle, it would have auto-concurrency instead of an auto-conflict.

Clearly, an STG without internal transitions is deterministic if and only if it is without auto-concurrency and without dynamic auto-conflict; the latter is ensured if there are no structural auto-conflicts. Note that internal transitions enabled concurrently or being in conflict do not introduce auto-concurrency or -conflict.

Simulations are a well-known important device for proving language inclusion or equivalence. A *simulation from* N_1 *to* N_2 is a relation $\mathcal{S}$ between markings of N_1 and N_2 such that $(M_{N_1}, M_{N_2}) \in \mathcal{S}$ and for all $(M_1, M_2) \in \mathcal{S}$ and $M_1[t\rangle M_1'$ there is some M_2' with $M_2[l_1(t)\rangle\rangle M_2'$ and $(M_1', M_2') \in \mathcal{S}$. If such a simulation exists, then N_2 can go on simulating all signals of N_1 forever.

Theorem 2.2 *If there exists a simulation from* N_1 *to* N_2*, then* $L(N_1) \subseteq L(N_2)$.

Often, nets are considered to have the same behaviour if they are language equivalent. Another, more detailed behaviour equivalence is bisimilarity. A relation $\mathcal{B}$ is a *bisimulation* between N_1 and N_2 if it is a simulation from N_1 to N_2 and $\mathcal{B}^{-1}$ is a simulation from N_2 to N_1. If such a bisimulation exists, we call the STGs *bisimilar*; intuitively, the STGs can work side by side such that in each stage each STG can simulate the signals of the other. For deterministic STGs, language equivalence and bisimulation coincide.

In the following definition of *parallel composition* $\|$, we will have to consider the distinction between input and output signals. The idea of parallel composition is that the composed systems run in parallel synchronizing on common signals. Since a system controls its outputs, we cannot allow a signal to be an output of more than one component; input signals, on the other hand, can be shared. An output signal of one component can be an input of one or several others, and in any case it is an output of the composition. A composition can also be ill-defined due to what e.g. Ebergen [Ebe92] calls computation interference; this is a semantic problem, and we will not consider it here, but later in the definition of correctness.

The parallel composition of STGs N_1 and N_2 is defined if $Out_1 \cap Out_2 = \emptyset$. Then, let $A = (In_1 \cup Out_1) \cap (In_2 \cup Out_2)$ be the set of common signals. If e.g. s is an output of N_1 and an input of N_2, then an occurrence of an edge $s\pm$ in N_1 is 'seen' by N_2, i.e. it must be accompanied by an occurrence of $s\pm$ in N_2. Since we do not know a priori which $s\pm$-labelled transition of N_2 will occur together with some $s\pm$-labelled transition of N_1, we have to allow for each possible pairing. Thus, the *parallel composition* $N = N_1 \parallel N_2$ is obtained from the disjoint union of N_1 and N_2 by combining each $s\pm$-labelled transition t_1 of N_1 with each $s\pm$-labelled transition t_2 from N_2 if $s \in A$. In the formal definition of parallel composition, $*$ is used as a dummy element, which is formally combined e.g. with those transitions that do not have their label in the synchronization set A. (We assume that $*$ is not a transition or a place of any net.) Thus, N is defined by

$$
\begin{aligned}
P &= P_1 \times \{*\} \cup \{*\} \times P_2 \\
T &= \{(t_1,t_2) \mid t_1 \in T_1, t_2 \in T_2, l_1(t_1) = l_2(t_2) \in A\{+,-\}\} \\
&\quad \cup \{(t_1,*) \mid t_1 \in T_1, l_1(t_1) \notin A\{+,-\}\} \\
&\quad \cup \{(*,t_2) \mid t_2 \in T_2, l_2(t_2) \notin A\{+,-\}\}
\end{aligned}
$$

$$
W((p_1,p_2),(t_1,t_2)) = \begin{cases} W_1(p_1,t_1) \text{ if } p_1 \in P_1,\ t_1 \in T_1 \\ \text{or} \\ W_2(p_2,t_2) \text{ if } p_2 \in P_2,\ t_2 \in T_2 \end{cases}
$$

$$
W((t_1,t_2),(p_1,p_2)) = \begin{cases} W_1(t_1,p_1) \text{ if } p_1 \in P_1,\ t_1 \in T_1 \\ \text{or} \\ W_2(t_2,p_2) \text{ if } p_2 \in P_2,\ t_2 \in T_2 \end{cases}
$$

$$
l((t_1,t_2)) = \begin{cases} l_1(t_1) & \text{if } t_1 \in T_1 \\ l_2(t_2) & \text{if } t_2 \in T_2 \end{cases}
$$

$$
M_N = M_{N_1} \dot\cup M_{N_2}, \text{ i.e. } M_N((p_1,p_2)) = \begin{cases} M_{N_1}(p_1) & \text{if } p_1 \in P_1 \\ M_{N_2}(p_2) & \text{if } p_2 \in P_2 \end{cases}
$$

$$
\begin{aligned}
In &= (In_1 \cup In_2) - (Out_1 \cup Out_2) \\
Out &= Out_1 \cup Out_2
\end{aligned}
$$

Clearly, one can consider the place set of the composition as the disjoint union of the place sets of the components; therefore, we can consider markings of the composition (regarded as multisets) as the disjoint union of markings of the components; the latter makes clear what we mean by the restriction $M\,|_{P_i}$ of a marking M of the composition.

We will denote a marking $M_1 \dot\cup M_2$ of the composition also by (M_1, M_2). By definition of $\|$, the firing $(M_1,M_2)[(t_1,t_2)\rangle(M_1',M_2')$ of N corresponds to the firings $M_i[t_i\rangle M_i'$ in N_i, $i = 1,2$; here, the firing of $*$ means that the empty transition sequence fires. Therefore, all reachable markings of N have the form (M_1,M_2), where M_i is a reachable marking of N_i, $i = 1,2$.

If the components do not have internal transitions, then also their composition has none. To see that N is deterministic if N_1 and N_2 are, consider different transitions (t_1,t_2) and (t_1',t_2') with the same label that are enabled under the reachable marking (M_1,M_2). The transitions differ in at least one component, say the first, and since it cannot be the case that t_1 is a transition while $t_1' = *$ (then we would have $l((t_1,t_2)) \in In_1\{+,-\} \cup Out_1\{+,-\}$ but $l((t_1',t_2')) \notin In_1\{+,-\} \cup Out_1\{+,-\}$), t_1 and t_1' are different transitions with the same label enabled under the reachable marking M_1, which contradicts that N_1 is deterministic. But note that N might have structural auto-conflicts even if none of the N_i has.

It should be clear that, up to isomorphism, composition is associative and commutative. Therefore, we can define the parallel composition of a family (or collection) $(C_i)_{i\in I}$ of STGs as $\|_{i\in I}\, C_i$, provided that no signal is an output signal of more than one of the C_i. We will also denote the markings of such a composition by $(M_1,\ldots,M_n)$ if M_i is a marking of C_i for $i \in I = \{1,...,n\}$. We close with a result on consistency.

Proposition 2.3 *If $(C_i)_{i\in I}$ is a family of consistent (deterministic resp.) STGs and $C = \|_{i\in I} C_i$ is defined, then C is also consistent (deterministic resp.).*

Proof: For determinism, see above. Take a signal s of C and w.l.o.g. let s be a signal of C_1. For $v \in L(C)$, let v_1 be the projection of v to $In_1 \cup Out_1$; one can show that $v_1 \in L(C_1)$. Therefore, the edges of s alternate along each such v_1 appropriately, and thus they do so on each $v \in L(C)$. □

3 Transition Contraction and Redundant Places

We now introduce and study transition contraction (see e.g. [And83] for an early reference), which will be most important in our decomposition procedure. We repeat largely from [VW02], where further discussions can be found, but we will also add some results which are important for our generalization of the decomposition method. In the following definition, we add the notion of a new conflict pair.

Definition 3.1 Let N be an STG and $t \in T$ with $W(.,t), W(t,.) \in \{0,1\}$, ${}^\bullet t \cap t^\bullet = \emptyset$ and $l(t) = \lambda$. We define the *t-contraction* $\overline{N}$ of N by

$$\begin{aligned}
\overline{P} &= \{(p,*) \mid p \in P - ({}^\bullet t \cup t^\bullet)\} \\
&\quad \cup \{(p,p') \mid p \in {}^\bullet t, p' \in t^\bullet\} \\
\overline{T} &= T - \{t\}
\end{aligned}$$

$$\begin{aligned}
\overline{W}((p,p'),t_1) &= W(p,t_1) + W(p',t_1) \\
\overline{W}(t_1,(p,p')) &= W(t_1,p) + W(t_1,p') \\
\overline{l} &= l|_{\overline{T}} \\
M_{\overline{N}}((p,p')) &= M_N(p) + M_N(p') \\
\overline{In} = In & \qquad \overline{Out} = Out
\end{aligned}$$

In this definition, $* \notin P \cup T$ is a dummy element; we assume $W(*,t_1) = W(t_1,*) = M_N(*) = 0$.

We say that the markings M of N and $\overline{M}$ of $\overline{N}$ satisfy the *marking equality* if for all $(p,p') \in \overline{P}$

$$\overline{M}((p,p')) = M(p) + M(p').$$

For two different transitions t_1, t_2 with $t_1 \neq t \neq t_2$, we call $\{t_1,t_2\}$ a *new conflict pair* whenever ${}^\bullet t \cap {}^\bullet t_1 \neq \emptyset$ and $t^\bullet \cap {}^\bullet t_2 \neq \emptyset$ in N. □

Note that, in general, $\overline{N}$ might fail to be consistent; we will have to study for which cases it is consistent, since this is usually required.

Figure 2 (a) shows a part of a net and the result when the internal transition is contracted. In many cases, the preset or the postset of the contracted transition has only one element, and then the result of the contraction looks much easier as e.g. in Figure 2 (b). Here, the b+- and the c+-labelled transition form a new conflict pair; note that this is also true, if they already have a common place (not drawn) in their presets in N.

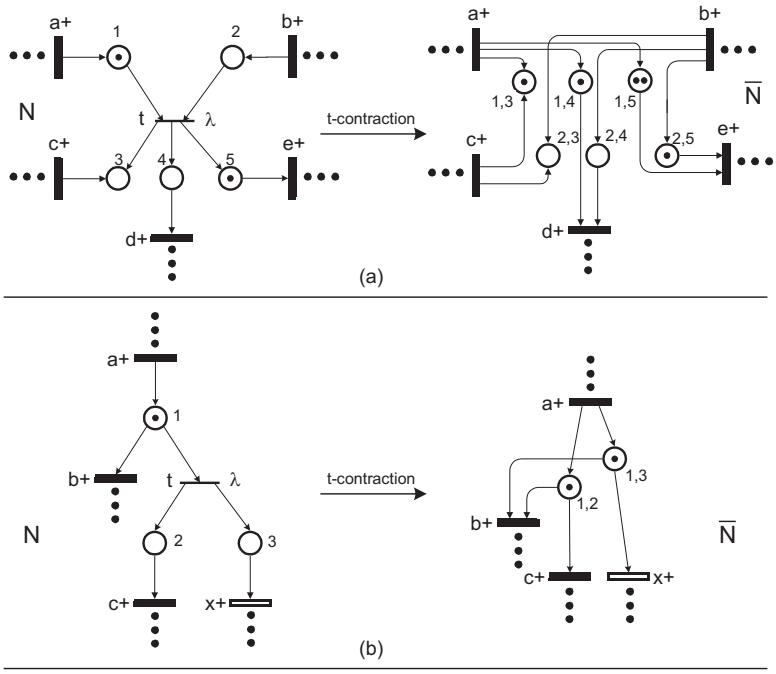


Figure 2

For the rest of this section, we fix an STG N with a transition t satisfying the requirements of Definition 3.1 and denote its t-contraction by $\overline{N}$. Furthermore, we define the relation $\mathcal{B}$ as $\{(M, \overline{M}) \mid M \text{ and } \overline{M} \text{ satisfy the marking equality}\}$.

The first theorem will show that contraction preserves behaviour in a weak sense, i.e. that all the traces of N will still be possible in $\overline{N}$. We begin with an important lemma which relates the transition firing of a net before and after contraction in two markings related by the marking equality.

Lemma 3.2 *Let M and $\overline{M}$ be markings of N and $\overline{N}$ satisfying the marking equality.*

1. *$M[t\rangle M'$ implies $\overline{M}((p,p')) = M'(p) + M'(p')$.*
2. *If $M[t_1\rangle M'$ for $t_1 \neq t$, then $\overline{M}[t_1\rangle\overline{M'}$, and M' and $\overline{M'}$ satisfy the marking equality.*
3. *$M[v\rangle\rangle M_1$ implies $\overline{M}[v\rangle\rangle\overline{M}_1$ such that also M_1 and $\overline{M_1}$ satisfy the marking equality.*

For the refined method we will present in this paper, we need the second part of the following theorem, for which we introduce a new notion:

Definition 3.3 An operation transforming an STG S into an STG S' is *auto-cc-preserving* if the following holds: if there are equally labelled observable transitions enabled under the same reachable marking of S, then the same is true for S'. (In other words, if S has auto-concurrency or an auto-conflict, then so has S'.) □

Theorem 3.4 *1. $\mathcal{B}$ is a simulation from N to $\overline{N}$; in particular, $L(N) \subseteq L(\overline{N})$.*

2. Transition contraction is auto-cc-preserving.

Proof: The first part follows from the last part of Lemma 3.2, since the initial markings satisfy the marking equality. The second part follows from the second part of Lemma 3.2. □

The next two results show that under additional assumptions contraction preserves behaviour in a stronger sense, i.e. the t-contraction is bisimilar or at least language-equivalent in these cases. In both theorems, the fourth and the new fifth part (the latter being implied by the new third part) are what is really needed for the remainder of the paper. A further comment on the assumptions can be found after Theorem 3.6.

Theorem 3.5 *Assume that* $({}^\bullet t)^\bullet \subseteq \{t\}$. *Then:*

1. $\mathcal{B}$ *is a bisimulation from* N *to* $\overline{N}$.
2. *If* t_1 *and* t_2 *with* $t_1 \neq t_2$ *are concurrently enabled under a reachable marking* $\overline{M}$ *of* $\overline{N}$, *then there is a reachable marking* M' *of* N *that satisfies the marking equality with* $\overline{M}$ *and also enables* t_1 *and* t_2 *concurrently.*
3. *If* t_1 *and* t_2 *with* $t_1 \neq t_2$ *are in conflict under a reachable marking* $\overline{M}$ *of* $\overline{N}$, *then they are also in conflict under some reachable marking* M' *of* N *satisfying the marking equality with* $\overline{M}$.
4. *The contraction preserves boundedness and freedom from auto-concurrency.*
5. *If* N *is free of dynamic auto-conflicts, then so is* $\overline{N}$.

Proof: First observe that the last two parts follow from the other three, hence we concentrate on these. In particular, if each reachable marking of $\overline{N}$ satisfies the marking equality with some reachable marking of N and N is k-bounded, then $\overline{N}$ is $2k$-bounded.

If ${}^\bullet t = \emptyset$, then $\overline{P} = P - t^\bullet$ and the marking $\overline{M}$ satisfying the marking equality with some marking M of N is given by $\overline{M} = M|_{\overline{P}}$. Since t can put arbitrarily many tokens onto $t^\bullet$, the three claims are quite easy to see; hence, let ${}^\bullet t \neq \emptyset$.

$\mathcal{B}$ is a simulation by Lemma 3.2, hence we only have to show that $\mathcal{B}^{-1}$ is a simulation, too. Let $(M, \overline{M}) \in \mathcal{B}$ and $\overline{M}[t_1\rangle M_1$. Firing t under M as often as possible gives a marking M' that still satisfies the marking equality with $\overline{M}$ by Lemma 3.2 and $M'(p_0) = 0$ for some $p_0 \in {}^\bullet t$. (For the latter, we use $W(p_0, t) = 1$ according to the precondition on t in Definition 3.1.) We check the places $p \in P$ to see that M' enables t_1:

$p \notin {}^\bullet t \cup t^\bullet$: $W(p, t_1) = \overline{W}((p, *), t_1) \leq \overline{M}((p, *)) = M'(p)$
$p \in {}^\bullet t$: $W(p, t_1) = 0$ by assumption
$p \in t^\bullet$: $W(p, t_1) = W(p_0, t_1) + W(p, t_1) = \overline{W}((p_0, p), t_1) \leq \overline{M}((p_0, p))$
$= M'(p_0) + M'(p) = M'(p)$

Now we have $M'[t_1\rangle M_1'$ for some M_1' and $(M_1', M_1) \in \mathcal{B}$ by Lemma 3.2. Since a sequence of t's followed by t_1 has the same label as just t_1, we have shown the first part.

For the second part, one finds a pair $(M, \overline{M}) \in \mathcal{B}$ and then constructs M' similarly as above, and also the check that M' enables t_1 and t_2 concurrently is very similar; e.g. for $p \notin {}^\bullet t \cup t^\bullet$, we have:
$W(p,t_1) + W(p,t_2) = \overline{W}((p,*),t_1) + \overline{W}((p,*),t_2) \leq \overline{M}((p,*)) = M'(p)$

For the third part, assume that there is a conflict between t_1 and t_2 under $\overline{M}$. First of all, t_1 and t_2 are enabled under $\overline{M}$, and hence also under M' as above. Now assume the conflict is due to (p', p), i.e. $\overline{W}((p',p),t_1) + \overline{W}((p',p),t_2) > \overline{M}((p',p))$; observing $W(p',t_i) = 0$ for $i = 1,2$ by assumption, we get $W(p,t_1) + W(p,t_2) > M'(p') + M'(p) \geq M'(p)$. □

Theorem 3.6 *Assume that ${}^\bullet(t^\bullet) = \{t\}$; in particular, $t^\bullet \neq \emptyset$. Further, assume that $\exists p_0 \in t^\bullet : \ M_N(p_0) = 0$; then:*

1. *$\{(\overline{M}, M) \in \mathcal{B}^{-1} \mid \exists q_0 \in t^\bullet : \ M(q_0) = 0\}$ is a simulation from $\overline{N}$ to N; N and $\overline{N}$ are language equivalent.*

2. *If t_1 and t_2 with $t_1 \neq t_2$ are concurrently enabled under a reachable marking $\overline{M}$ of $\overline{N}$, then there is a reachable marking M' of N that satisfies the marking equality with $\overline{M}$ and also enables t_1 and t_2 concurrently.*

3. *If t_1 and t_2 with $t_1 \neq t_2$ are in conflict under a reachable marking $\overline{M}$ of $\overline{N}$, then they are also in conflict under some reachable marking M' of N satisfying the marking equality with $\overline{M}$ or $\{t_1, t_2\}$ is a new conflict pair .*

4. *The contraction preserves boundedness and freedom from auto-concurrency.*

5. *If N is free of dynamic auto-conflicts, but $\overline{N}$ is not due to some t_1 and t_2, then $\{t_1, t_2\}$ is a new conflict pair.*

Proof: First observe that last two parts follow from the other three, hence we concentrate on these. In particular, each reachable marking of $\overline{N}$ satisfies the marking equality with some reachable marking of N; hence, if N is k-bounded, then $\overline{N}$ is $2k$-bounded.

To show the first part, observe that the initial markings are related by hypothesis. Now assume $(\overline{M}, M)$ is in the given relation, $q_0 \in t^\bullet$ with $M(q_0) = 0$, and $\overline{M}[t_1\rangle M_1$.

We choose $p_1 \in t^\bullet$ such that $m_1 = W(p_1,t_1) - M(p_1)$ is maximal; since $W(q_0,t_1) - M(q_0) = W(q_0,t_1) \geq 0$, m_1 is not negative. We check that t can fire m_1 times under M: for all $p \in {}^\bullet t$, we have $M(p) + M(p_1) = \overline{M}((p,p_1)) \geq \overline{W}((p,p_1),t_1) = W(p,t_1) + W(p_1,t_1)$, and thus $M(p) \geq W(p_1,t_1) - M(p_1) + W(p,t_1) \geq m_1$. Firing t under M m_1 times gives a marking M', which satisfies the marking equality with $\overline{M}$ by Lemma 3.2. By choice of p_1, we have: $(*)$ $M'(p_1) = W(p_1,t_1)$ and $M'(p) \geq W(p,t_1)$ for all $p \in t^\bullet$.

We check that t_1 is enabled under M' by considering all places p:

$p \notin {}^\bullet t \cup t^\bullet$: $W(p,t_1) = \overline{W}((p,*),t_1) \leq \overline{M}((p,*)) = M'(p)$
$p \in t^\bullet$: $M'(p) \geq W(p,t_1)$ by $(*)$
$p \in {}^\bullet t$: $W(p,t_1) + W(p_1,t_1) = \overline{W}((p,p_1),t_1) \leq \overline{M}((p,p_1)) = M'(p) + M'(p_1)$
$= M'(p) + W(p_1,t_1)$ by $(*)$, hence $W(p,t_1) \leq M'(p)$

Now we have $M'[t_1\rangle M_1'$ for some M_1'. Since $W(t_1,p_1) = 0$ by hypothesis, we have $M_1'(p_1) = 0$ by $(*)$. Therefore (M_1, M_1') is in the given relation by Lemma 3.2. As above, since a sequence of t's followed by t_1 has the same label as just t_1, we have shown the first claim.

Language equivalence follows since, together with Theorem 3.4, we have simulations in both directions.

The second part can be shown in a similar way. If $\overline{M}$ is reachable, it is related to some reachable M by the simulation of the first part; let $q_0 \in t^\bullet$ with $M(q_0) = 0$.

We construct M' as above and check that M' enables t_1 and t_2 concurrently as above by adding in the above argument to every atomic term containing t_1 an analogous term containing t_2. E.g. we choose $p_1 \in t^\bullet$ such that $m_1 = W(p_1,t_1)+W(p_1,t_2)-M(p_1)$ is maximal; due to q_0, m_1 is not negative.

For the third part, we find a pair $(\overline{M}, M)$ in the given relation, m_1 for t_1 as above and m_2 for t_2 analogously. We assume that $\{t_1,t_2\}$ is not a new conflict pair.

Now we distinguish several cases. First assume that $({}^\bullet t_1 \cup {}^\bullet t_2) \cap t^\bullet = \emptyset$. In this case, we have $m_1 = m_2 = 0$ and the respective M' (which equals M) enables t_1 and t_2 as above. If the conflict under $\overline{M}$ is due to some $(p,*)$, then $W(p,t_1) + W(p,t_2) = \overline{W}((p,*),t_1) + \overline{W}((p,*),t_2) > \overline{M}((p,*)) = M'(p)$. If the conflict is due to some (p,p'), then we have due to $W(p',t_1) = W(p',t_2) = 0$ that $W(p,t_1) + W(p,t_2) = \overline{W}((p,p'),t_1) + \overline{W}((p,p'),t_2) > \overline{M}((p,p')) \geq M'(p)$. In any case, t_1 and t_2 are in conflict under M'.

Second, we assume that, say, ${}^\bullet t_1 \cap t^\bullet \neq \emptyset$ while ${}^\bullet t_2 \cap t^\bullet = \emptyset$; due to assumption, we also have ${}^\bullet t_2 \cap {}^\bullet t = \emptyset$. We construct M' due to m_1, hence M' enables t_1 as above; for t_2, we only have to check the $p \notin {}^\bullet t \cup t^\bullet$, which works as in the proof for the first part. Thus, M' also enables t_2; the conflict under $\overline{M}$ must be due to some $(p,*)$, which implies a conflict under M' as in the first case.

In the third case, we have ${}^\bullet t_1 \cap t^\bullet \neq \emptyset$ and ${}^\bullet t_2 \cap t^\bullet \neq \emptyset$; thus, by assumption, we have ${}^\bullet t_1 \cap {}^\bullet t = \emptyset$ and ${}^\bullet t_2 \cap {}^\bullet t = \emptyset$. W.l.o.g. we have $m_1 \geq m_2$, and we construct M' from m_1 as above. Thus, M' enables t_1. For the enabledness of t_2, we only have to check $p \notin {}^\bullet t \cup t^\bullet$ (works as before) and $p \in t^\bullet$; for the latter, firing t m_2 times would already have ensured $M'(p) \geq W(p,t_2)$ – and firing t more often does not make this wrong; hence, M' also enables t_2. If the conflict under $\overline{M}$ is due to some $(p,*)$, we derive a conflict under M' as in the first case. If the conflict is due to some (p,p'), then we have due to $W(p,t_1) = W(p,t_2) = 0$ that $W(p',t_1) + W(p',t_2) = \overline{W}((p,p'),t_1) + \overline{W}((p,p'),t_2) > \overline{M}((p,p')) \geq M'(p')$; also in this case, t_1 and t_2 are in conflict under M'. □

If the preconditions of Definition 3.1 and Theorem 3.5 or 3.6 are satisfied, then we call the contraction of t *secure (of type 1 or type 2)*.

The last two theorems show that, for a secure contraction of t, each reachable marking of $\overline{N}$ can be determined from a reachable marking of N via the marking

equality. Furthermore, a marking M of N enabling t will be related to the same marking of $\overline{N}$ as the marking M' with $M[t\rangle M'$. This implies that the contracted net has at most as many reachable markings as N, and usually less.

To explain the definition of a secure contraction, assume that there are transitions t_1 and t_2 with $t_1 \neq t \neq t_2$, $p_1 \in {}^\bullet t_1 \cap {}^\bullet t$ and $p_2 \in t_2{}^\bullet \cap t^\bullet$. After contracting t, t_2 can put a token onto (p_1, p_2) and t_1 can take it, such that, intuitively speaking, the token flows from the postset of t back to the preset; clearly, this *backfiring* can lead to completely new behaviour in $\overline{N}$, such that on the one hand we cannot expect language equivalence, while on the other hand auto-concurrency could be introduced.

As to our results about the introduction of auto-conflicts, observe that this would be rather trivial for structural auto-conflicts: If $\overline{N}$ obtained from N by contraction has a structural auto-conflict, while N has not, then certainly this is due to a new conflict pair – and this holds for an arbitrary contraction, secure or not. For dynamic auto-conflicts as treated in the above theorems, the situation is different. Figure 3 shows an STG N and the STG $\overline{N}$ obtained by a non-secure contraction: while N is actually *dead* (i.e. no transition can fire), $\overline{N}$ has a dynamic auto-conflict.

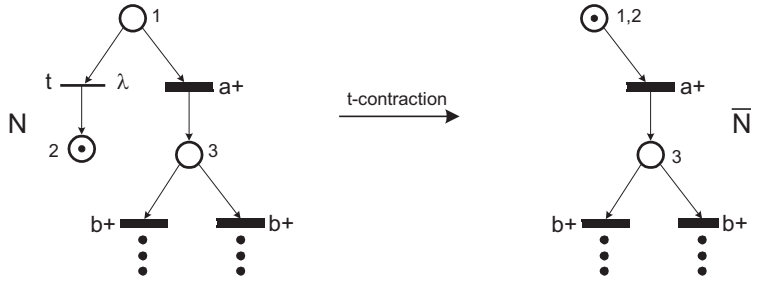


Figure 3

We call an operation on STGs *consistency-preserving*, if it turns a consistent STG into one that is consistent again. With this notion, we have the following corollary to Lemma 2.1 and Theorems 3.5 and 3.6.

Corollary 3.7 *Secure transition contractions are consistency-preserving.*

We conclude this section by defining redundant places; the deletion of such a place (including the incident arcs) is another operation that can be used in our decomposition algorithm. A place p of an STG N is (structurally) *redundant* (see e.g. [Ber87]) if there is a set of places Q with $p \notin Q$, a valuation $V : Q \cup \{p\} \to \mathbb{N}$ and some $c \in \mathbb{N}_0$ which satisfy the following properties for all transitions t:

- $V(p)M_N(p) - \sum_{q \in Q} V(q)M_N(q) = c$
- $V(p)(W(t,p) - W(p,t)) - \sum_{q \in Q} V(q)(W(t,q) - W(q,t)) \geq 0$
- $V(p)W(p,t) - \sum_{q \in Q} V(q)W(q,t) \leq c$

The first two items ensure that p is something like a linear combination of the places in Q with factors $V(q)/V(p)$. Indeed, for the case $c = 0$, the first item says that p is such a combination initially; the second item, in the case of equality, says that this relationship is preserved when firing any transition. The proof that p is indeed redundant argues that the valuated token number of p is at least c larger than the valuated token sum on Q for all reachable markings, while the third item says that each transition or at least each output transition needs at most c 'valuated tokens' more from p than from the places in Q; this shows that for the enabling of a transition the presence or absence of p does not matter. Therefore, the deletion of a redundant place in N turns each reachable marking of N into one of the transformed STG that enables the same transitions, hence the deletion gives a bisimilar STG – which is consistent if N is.

A special case of a redundant place is a *loop-only place*, i.e. a marked place p such that p and t form a a loop with arcs of weight 1 for all $t \in {}^\bullet p \cup p^\bullet$. Another simple case is that of a duplicate: place p is an (extended) *duplicate* of place q, if for all transitions t $W(t,p) = W(t,q)$, $W(p,t) = W(q,t)$ and $M_N(p) \geq M_N(q)$.

4 Decomposing a Signal Transition Graph

4.1 Correctness Definition

For this section, we assume that we are given a fixed STG N as a specification of some desired behaviour. Our aim is to decompose it into a collection of components $(C_i)_{i\in I}$ that together implement the specified behaviour; in particular, this should help to avoid the state explosion problem, and therefore we have to avoid the complete state exploration for N.

In particular in the area of circuit design, it seems most often to be the case that, if an input or output is specified, then its effects are specified without any choices; therefore, we assume that N is *deterministic*. In contrast to [VW02], we do *not* require N to be free of *structural auto-conflicts*. As in [VW02], we will concentrate on the construction of components that are also deterministic.

The first new observation of the present paper concerns the following: it seems to be convenient to allow λ-transitions in N. The understanding of such so-called *dummy transitions* is that they do not represent state changes, i.e. what really matters about N is just its language [J. Cortadella, *priv. comm.*]; hence, when synthesizing a circuit via the reachability graph of N, one can remove the resulting λ-arcs in the reachability graph by well-known automata-theoretic methods. The problem is that we want to avoid the construction of this graph.

With the results of the previous section, we can instead contract the dummy transitions provided these contractions are secure. Since such contractions preserve the language, we can just as well work with the resulting STG if it is free of dynamic auto-conflicts.

Besides determinism, it is further assumed in [VW02] that N is free of *structural input/output conflicts*; this ensures that there are no dynamic input/output conflicts, which are very hard to implement, since the input, which is under the control of the

environment, might occur at roughly the same time as the output, which is under the control of the system, and can therefore not prevent the output as specified; technically, this may even lead to malfunction. But in the literature, we have found an example with a structural input/output conflict, which is even also a dynamic one; cf. stg_blunno in Section 6. We take this as motivation to generalize the approach of [VW02] to STGs with input/output conflicts; but given the problems such conflicts can create, we still regard it as important that our method does not introduce any new input/output conflicts. In applications, N will be bounded and most often even safe; but our results also hold in the general case.

We first repeat the definition when a collection of components is a correct implementation of N; for a detailed explanation and a discussion of related work, we refer to [VW02]. We will additionally show that the components our algorithm generates are consistent if N is.

Definition 4.1 A collection of deterministic components $(C_i)_{i\in I}$ is a *correct decomposition* or a *correct implementation* of a deterministic STG N, if the parallel composition C of the C_i is defined, $In_C \subseteq In_N$, $Out_C \subseteq Out_N$ and there is a relation $\mathcal{B}$ between the markings of N and those of C with the following properties.

1. $(M_N, M_C) \in \mathcal{B}$
2. For all $(M, M') \in \mathcal{B}$, we have:
   (a) If $a \in In_N$ and $M[a\pm\rangle M_1$, then either $a \in In_C$ and $M'[a\pm\rangle M_1'$ and $(M_1, M_1') \in \mathcal{B}$ for some M_1' or $a \notin In_C$ and $(M_1, M') \in \mathcal{B}$.
   (b) If $x \in Out_N$ and $M[x\pm\rangle M_1$, then $M'[x\pm\rangle M_1'$ and $(M_1, M_1') \in \mathcal{B}$ for some M_1'.
   (c) If $x \in Out_C$ and $M'[x\pm\rangle M_1'$, then $M[x\pm\rangle M_1$ and $(M_1, M_1') \in \mathcal{B}$ for some M_1.
   (d) If $x \in Out_i$ for some $i \in I$ and $M'|_{P_i}[x\pm\rangle$, then $M'[x\pm\rangle$. (no *computation interference*)

Here, and whenever we have a collection $(C_i)_{i\in I}$ in the following, P_i stands for P_{C_i}, Out_i for Out_{C_i} etc. □

The important features are the following. We allow C to have fewer input and output signals than N; this is possible for input signals which are irrelevant for producing the right outputs and for outputs that actually never have to be produced. (In fact, our algorithm only produces C with $Out_C = Out_N$.) There is no clause requiring a match for inputs of C; this implies that N and C are not necessarily language equivalent, as e.g. required in [Chu87a, Chu87b]; see [VW02] for a discussion why this is adequate and useful. Clause (d) is important for a correct functioning; if it is violated, then component C_i could produce some output that some other component is not ready to accept, which could lead to malfunction of this other component. Finally, it should be pointed out that the chosen style of this definition is technically useful in the correctness proof.

We repeat from [VW02], that it would also be possible to require in (b) only $M'[y\pm\rangle\rangle$ for some $y \in Out_C$: (c) would ensure that this output is specified and the ensuing behaviour matches the specification; but this clause would only say that, in case of *several* specified outputs, at least *one* will be performed. Whether the others are possible or not cannot be observed, since once an output is performed, it cannot be checked whether others had been possible as well; this view is e.g. taken in [Seg93]. Our decomposition algorithm guarantees the stronger form of correctness given in clause (b) above.

In the next subsection, we describe our decomposition algorithm, which uses what we call totally admissible operations. In the third subsection, we prove that it indeed produces correct components, and in the fourth, we show that certain contractions as well as certain place and transition deletions are totally admissible.

4.2 The Decomposition Algorithm

We start with a rough description of our algorithm, where the unknown notions will be described in this and the next subsection:

Given are a deterministic STG N (possibly obtained by securely contracting dummy transitions as explained above) as a specification, and a feasible partition of its signals. The algorithm constructs from these an initial decomposition $(C_i)_{i\in I}$. Then it repeatedly applies a totally admissible operation (from some list of such operations) or backtracking to one of the C_i after the other until it does not contain any λ-transitions anymore.

To initialize the algorithm, one has to choose a feasible partition of the signals of N; our condition (C1) for such a partition is different from [VW02], since we allow input/output-conflicts in this paper. A *feasible partition* is a family $(In_i, Out_i)_{i\in I}$ for some set I such that the sets Out_i, $i \in I$, are a partition of Out_N and for each $i \in I$ we have $In_i \subseteq In_N \cup Out_N \setminus Out_i$, and furthermore:

(C1) If signal s and output signal x of N are in structural conflict, then $x \in Out_i$ implies $s \in In_i$ if $s \in In$ and $s \in Out_i$ if $s \in Out$ for each $i \in I$.

The rationale for this relies on the above discussion of input/output conflicts: clearly, a component responsible for output signal x must at least 'see' any signal that could be in dynamic conflict with x in N; if such a signal is an output as well, the component should also produce it, because otherwise we would have a new input/output conflict.

(C2) If there are $t, t' \in T_N$ with $t^\bullet \cap {}^\bullet t' \neq \emptyset$ and the signal of t' is in Out_i for some $i \in I$, then the signal of t is in $In_i \cup Out_i$. (The latter signal *gives concession* to the first one. It might be in In_i even if it belongs to Out; in this case, it will be produced by some other component, and the ith component just listens to it.)

For a feasible partition, the *initial decomposition* is $(C_i)_{i\in I}$, where each *initial component* $C_i = (P, T, W, l_i, M_N, In_i, Out_i)$ is a copy of N except for the labelling and the signal classification; $l_i(t) = l(t)$ if the signal of t is in $In_i \cup Out_i$ and $l_i(t) = \lambda$

otherwise. For reasons that will become clear later on in the correctness proof, we call these λ-transitions *divining* transitions.

Observe that all the initial components are free of dynamic auto-conflicts and auto-concurrency, since N is deterministic. Furthermore, if one of them has a structural, dynamic resp., input/output-conflict, then already N has one for the same signals.

The main idea of the algorithm is now to remove the λ-transitions using secure transition contractions. Unfortunately, the algorithm could get stuck, e.g. when no secure contraction is applicable although there are still λ-transitions left; therefore, it can also do the following:

- **Backtracking:** backtracking applied to some C_i and some signal $s \notin In_i \cup Out_i$ adds s to In_i and replaces C_i by the respective new initial component.

  Observe that this modifies the feasible partition in such a way that the resulting partition is feasible again; in particular, C_i already has all signals that are in structural conflict to an output signal of C_i.

  Backtracking undoes all the totally admissible operations that have already been performed on C_i. In many cases, it will be possible to perform some of these also on the new initial component; hence, we will study in the future how to implement backtracking such that not always all the operations are undone.

Here is the list of totally admissible operations we use in this paper. The first is the heart of the algorithm, and should definitely be on any such list; the other two improve the results. In the context of STG-decomposition, RedPD was suggested for the first time in [VW02], while RedTD is new; further operations may turn up in the future.

- **SecTC:** Perform a secure transition contraction to some t of some C_i, provided this gives an STG without dynamic auto-conflicts.

  If there is such a conflict, the algorithm will perform backtracking on C_i and the signal of t next – but this should not be seen as part of SecTC. The rationale for this is: the dynamic auto-conflict shows that C_i should better know about the signal edge labelling t in N in order to decide which of the two equally labelled transitions to fire.

  It is not obvious how to detect dynamic auto-conflicts without building the reachability graph; one way to avoid such conflicts (chosen in [VW02]) is to apply SecTC only if no structural auto-conflict is created. New results of this paper also allow other possibilities, and we will discuss them at the end of Subsection 4.5.

- **RedPD:** Delete a redundant place in some C_i.

- **RedTD:** Delete a divining transition t in some C_i, where either each place $p \in {}^\bullet t \cup t^\bullet$ forms a loop with t with two arcs of the same weight (t is a *loop-only transition*) or some other divining transition has arcs to and from the same places with the same weight as t (which is a *duplicate transition*).

The latter two operations seem rather trivial since they clearly do not change the behaviour of the respective C_i. Still, both of them have turned out to be essential for good results in some cases; furthermore, both operations change the net structure, and since our proof below refers to the net structure, some care should be taken when adding them to the list. In fact, we structure the presentation of the algorithm and of the proof differently from [VW02], partly because it was not possible to add RedTD directly to the algorithm as described in [VW02].

There, operations are from the beginning restricted to SecTC and RedPD, while backtracking was explicitly described in the algorithm as exception handling in case of auto-conflicts. Our new presentation treats this as an implementation detail, as a strategy for choosing the next operation; this seems much better suited for adding further operations.

Note that the algorithm is nondeterministic; as an extreme case, one could choose to apply backtracking only, which clearly does not give a useful result. A sensible implementation only applies backtracking if there is no alternative; but also then, backtracking might be applied so often that some resulting C_i becomes too large. In an extreme case, some C_i could be equal to N except for the classification of signals as inputs or outputs. But even in such a case, the result might be useful if e.g. an arbiter has been split off; see [VW02] for such an example. This is in fact a very relevant case of finding a library element that can be employed.

Still, in some cases the algorithm might fail to produce something useful. This is actually not surprising, since one cannot expect to fight state explosion successfully in all cases. All one can hope for are reasonable results reasonably often; in the examples we have checked, the algorithm most often performed quite well.

It should also be remarked that the result of our algorithm is not uniquely determined, even in case of a sensible implementation. [VW02] shows an example where the order of trying to apply transition contractions decides for which signal backtracking (due to an auto-conflict) will be performed; this results in two different decompositions. This issue deserves further investigations, in particular since we want to find components with small reachability graphs.

4.3 The Correctness Proof I

We will first discuss the issue of termination. For the definition of a totally admissible operation, one has to fix a function (possibly referring to N) from STGs with signals in $In \cup Out$ into some set with a well-founded ordering as a *termination function*; we choose here the function that gives for such an STG C the triple (sc, tc, pc), where sc is $|In \cup Out| - |In_C \cup Out_C|$ (the number of signals missing in C), tc is the number of transitions, and pc is the number of places of C. We order such triples lexicographically according to the standard $\leq$ on natural numbers.

When extending the above list of operations in future work, it might be necessary to modify this termination function; in this sense, our approach should be seen as parametric with the termination function as parameter.

A *totally admissible operation* is an admissible operation that applied to an STG with signals in $In \cup Out$ does not change the sets of input and output signals and makes the value of the termination function smaller.

As long as there is still some divining transition in some C_i, an operation can be applied – we can always choose backtracking. When backtracking is applied, the number of signals missing in C_i goes down and so the value of the termination function decreases for all operations applied in the algorithm; hence, the algorithm terminates.

Backtracking changes the feasible partition; had we started out with this new partition, the operations performed on some other C_j before the backtracking would give the same result. Further, we have already concluded that backtracking can only be applied finitely we often. Thus, the result of the algorithm can be seen as being obtained (from a modified feasible partition) by totally admissible operations alone (without any backtracking), and it suffices to show correctness for such a case. We will do so in the next subsection.

The precise definition of an admissible operation is tuned to this correctness proof and rather technical; hence, we will postpone it for the moment. Instead, we collect the correctness results we will obtain in the rest of this section. For this, we need one further notion.

Definition 4.2 An operation transforming some STG S to an STG S' does *not introduce io-conflicts* if, for any structural, dynamic resp., input/output-conflict in S', S already has one for the same signals. □

Theorem 4.3 *1. The decomposition algorithm terminates for each deterministic STG N; the resulting components $(C_i)_{i\in I}$ are deterministic and form a correct implementation of N.*

2. If the decomposition algorithm uses only the operations presented in this paper, we have additionally: if N is consistent, then so are the components $(C_i)_{i\in I}$; if some of the components has a structural, dynamic resp., input/output-conflict then N has one for the same signals.

Proof: 1. This follows from our above considerations and from Lemma 4.5.2 and Theorem 4.8 below.

2. First, assume that N is consistent. Then, each initial component is also consistent, since its language is the projection of $L(N)$ to the signals of this component. In Subsection 4.5, we will show that each of our totally admissible operations is consistency-preserving.

The second claim holds for the initial components due to (C1), since they have the same reachable markings as N. In Subsection 4.5, we will show that none of our totally admissible operations introduces io-conflicts. □

We will further show that each of our totally admissible operations is auto-cc-preserving. This and Theorems 3.5.5 and 3.6.5 will become relevant, when we discuss the implementation of SecTC at the end of the present section.

In the rest of the section, we will define admissible operations, show that they guarantee correctness, and prove that the operations on the list above are indeed admissible. It is easy to see that then they are also totally admissible: the number of missing signals is unchanged by all three operations; SecTC reduces the number of

transitions, which also holds for RedTD; RedPD leaves this number unchanged but reduces the number of places.

As in [VW02], our (slightly different) notion of an admissible operation structures our proofs, allows us to do the correctness proof in the next subsection only once instead of several times for each of the operations, and it supports a possible future reuse of this proof.

4.4 The Correctness Proof II

When we speak of applying an operation to an STG, this always implies in the following that all signals of this STG belong to $In \cup Out$. Each admissible operation will be pre-admissible in the following sense:

Definition 4.4 An operation is *pre-admissible* if, whenever applied to an STG without auto-concurrency and dynamic auto-conflicts and satisfying (a) and (b) below, it gives an STG satisfying these four properties again:

(a) There is no structural λ/output conflict.

(b) If t_2 is an output transition and ${t_1}^\bullet \cap {}^\bullet t_2 \neq \emptyset$, then t_1 is not an internal transition.

□

Two easy consequences of this definition are formulated in the following lemma, which will be needed when proving correctness formulated in Theorem 4.8.

Lemma 4.5 *1. At each stage of the algorithm, each C_i satisfies (a) and (b) of Definition 4.4 and is free of auto-concurrency and dynamic auto-conflicts.*

2. When the algorithm terminates, the resulting C_i and hence also C are deterministic.

Proof: The initial C_i satisfy (a): if an output transition of C_i is in structural conflict with another transition, then the signal of the latter is also a signal of C_i due to condition (C1) in the definition of a feasible partition; hence, the latter transition is not labelled λ. They satisfy (b) by condition (C2) of this definition. Furthermore, they are free of auto-concurrency and dynamic auto-conflicts since N is. Hence, the first claim follows from Definition 4.4.

Now the second claim follows from Part 1, since the algorithm only terminates when there are no divining transitions left. □

We now come to another central notion in our correctness proof; it is a variant of a bisimulation with an angelic treatment of internal transitions, and it is (like a loop invariant) needed to describe in what sense the intermediate stages of our algorithm are correct. If there is an internal transition in an initial C_i, then this corresponds to a signal edge of the system that this component does not 'see'; if we assume that by some angelic intervention (or by its capability to *divine*) such a transition, which is internal to C_i and not connected in any way to the outside, fires if the signal edge occurs, then the C_i together work as intended, and this sort of behaviour is captured

with an angelic bisimulation. For the final C_i, an angelic bisimulation guarantees correctness, since there are no internal transitions.

Clearly, if we synthesized a circuit for an intermediate C_i with internal transitions, we could not assume that the internal transitions of the circuit would behave well, i.e. in accordance with the angelic bisimulation. Hence, this is not a 'real-life' correctness notion; angelic correctness is just a mathematical tool for our proof.

We regard it as highly interesting that this kind of bisimulation is useful even though we do not assume any angelic nondeterminism in our correctness definition (4.1). In the future, we will study decompositions where components communicate with each other by signals that are internal to the implementation; these internal signals will certainly have to be of a different kind compared to the divining λ-transitions we study here.

Definition 4.6 A collection of components $(C_i)_{i\in I}$ is an *angelically correct decomposition* or *implementation* of a deterministic STG N, if the parallel composition C of the C_i is defined, $In_C \subseteq In_N$, $Out_C \subseteq Out_N$ and there is an *angelic bisimulation* relation $\mathcal{B}$ between the markings of N and those of C, i.e. $\mathcal{B}$ satisfies the following properties.

1. $(M_N, M_C) \in \mathcal{B}$
2. For all $(M, M') \in \mathcal{B}$, we have:
   (a) If $a \in In_N$ and $M[a\pm\rangle\rangle M_1$, then either $a \in In_C$ and $M'[a\pm\rangle\rangle M_1'$ and $(M_1, M_1') \in \mathcal{B}$ for some M_1' or $a \notin In_C$ and $M'[\lambda\rangle\rangle M_1'$ and $(M_1, M_1') \in \mathcal{B}$ for some M_1'.
   (b) If $x \in Out_N$ and $M[x\pm\rangle\rangle M_1$, then $M'[x\pm\rangle\rangle M_1'$ and $(M_1, M_1') \in \mathcal{B}$ for some M_1'.
   (c) If $x \in Out_i$ for some $i \in I$ and $M'\big|_{P_i}[x\pm\rangle\rangle$, then some M_1' and M_1 satisfy $M'[x\pm\rangle\rangle M_1'$, $M[x\pm\rangle\rangle M_1$ and $(M_1, M_1') \in \mathcal{B}$.

□

This definition looks very much like Definition 4.1; the differences are that here $[x\pm\rangle\rangle$ in C might involve additional λ-transitions besides an $x\pm$-labelled transition, that in 2(a) internal transitions are allowed to match an input of N that is not one of C, and that 2(c) is a combination of 4.1.2(c) and (d) and guarantees a matching only for some M_1' – this is an angelic part of the definition. It is also angelic that we do not require a match for the firing of only internal transitions in C.

We come to the final definition of an admissible operation, which leads to the correctness result.

Definition 4.7 We call a pre-admissible operation applied to some member of a family $(C_i)_{i\in I}$ that satisfies (a) and (b) of Definition 4.4 *admissible* if it preserves angelic correctness w.r.t. N. □

Theorem 4.8 *When the algorithm terminates, the resulting C_i are a correct decomposition of N.*

Proof: Due to $\mathcal{B} = \{(M, (M, \ldots, M) \mid M \text{ is reachable in } N\}$, the initial decomposition is angelically correct. The defining clauses for an angelic bisimulation are satisfied, since the firing of a transition t in N or in some C_i can be matched by firing all copies of this transition in N and all the C_i.

The only problem is case (c); assume $M'|_{P_i}[x\pm\rangle\rangle$. This image-firing involves an $x\pm$-labelled transition t. We have already convinced ourselves that C_i satisfies (b) of Definition 4.4; hence, only firing internal transitions cannot help to enable t, and thus t must already be enabled under $M|_{P_i}$. Therefore, the required match is obtained by letting all copies of t in N and the initial components fire.

Admissible operations preserve angelic correctness by definition, since the C_i always satisfy conditions (a) and (b) of Definition 4.4 by Lemma 4.5. Hence, the resulting C_i are an angelically correct decomposition of N.

Furthermore, the C_i and C are deterministic by Lemma 4.5. Therefore, (a), (b) and (c) of Definition 4.6 immediately give (a), (b) and (d) of Definition 4.1. Further, M_1' in (c) of 4.1 is uniquely determined by M' and $x\pm$ by determinism of C, thus it is the M_1' in (c) of 4.6 and therefore also (c) of 4.1 follows. □

It is useful to note the following:

Lemma 4.9 *Let a pre-admissible operation be given that, applied to some member of a family $(C_i)_{i\in I}$ satisfying (a) and (b) of Definition 4.4, transforms some C_j to a bisimilar $\overline{C_j}$ with the same input and output signals. Then the operation is admissible.*

Proof: Assume $(C_i)_{i\in I}$ is angelically correct for N due to the angelic bisimulation $\mathcal{B}$. Assume $j = 1$ and $J = I \setminus \{1\}$ such that we can write the markings of $C = \|_{i\in I}\, C_i$ as pairs (M', M'') with M' a marking of C_1 and M'' a marking of $C' = \|_{i\in J}\, C_i$; for $\overline{C} = \overline{C_1} \parallel \|_{i\in J}\, C_i$, the markings can be written as $(\overline{M'}, M'')$ with $\overline{M'}$ a marking of $\overline{C_1}$ and M'' a marking of C'. Let $\mathcal{R}$ be a bisimulation between C_1 and $\overline{C_1}$.

We define $\overline{\mathcal{B}}$ as $\{(M, (\overline{M'}, M'')) \mid (M, (M', M'')) \in \mathcal{B} \text{ and } (M', \overline{M'}) \in \mathcal{R}\}$. Clearly, $\overline{\mathcal{B}}$ satisfies Definition 4.6.1. For Definition 4.6.2, we check e.g. the case that $(M, (\overline{M'}, M'')) \in \overline{\mathcal{B}}$, $a \in In_N$, $M[a\pm\rangle\rangle M_1$ and $a \in In_{\overline{C}}$. In the match $(M', M'')[a\pm\rangle\rangle (M_1', M_1'')$, C_1 and C' perform a sequence of transitions with image $a\pm$ or λ; $\overline{C_1}$ can perform a sequence with the same image as C_1 from $\overline{M'}$ such that $(\overline{M'}, M'')[a\pm\rangle\rangle(\overline{M_1'}, M_1'')$ and $(M_1', \overline{M_1'}) \in \mathcal{R}$. Thus, $(M_1, (\overline{M_1'}, M_1'')) \in \overline{\mathcal{B}}$.

The other cases are similar, except for the case $x \in Out_1$ and $\overline{M'}[x\pm\rangle\rangle$. Since $(M', \overline{M'}) \in \mathcal{R}\}$, we also have $M'[x\pm\rangle\rangle$; then, we apply (c) for $\mathcal{B}$ to get (c) for $\overline{\mathcal{B}}$ as above. □

4.5 Admissible Operations

It remains to prove that all the operations on our list are admissible operations; we also check that each operation is consistency-preserving and that none of them introduces io-conflicts. The latter ensures, as already noted, that an STG N without structural, dynamic resp., input/output-conflicts (which we assume to be the standard case) is decomposed into component STGs that neither have structural, dynamic resp., input/output-conflicts.

By definition, a pre-admissible operation applied to an STG satisfying all four properties in Definition 4.4 gives an STG again satisfying these properties. It is a stronger result, if e.g. one of these properties is preserved on its own, like RedPD applied to an STG satisfying (a) of Definition 4.4 gives an STG again satisfying (a). Since such stronger preservation results might become useful in the future, we list them now and leave it to the reader to check this list while reading the proofs below: RedPD and RedTD preserve (a) and (b) of Definition 4.4 separately, while SecTC preserves their combination (i.e. their conjunction); all these operations preserve freeness from dynamic auto-conflicts, where in the last case this is by force – a transition contraction is not allowed if it introduces such a conflict; RedTD and SecTC preserve freeness from auto-concurrency, while RedPD preserves the combination freeness from dynamic auto-conflicts and from auto-concurrency.

We have already argued in Section 3, that the deletion of a redundant place turns each reachable marking of the original STG into one of the transformed STG that enables the same transitions, hence the deletion gives a bisimilar STG (from which consistency-preservation follows) and it is auto-cc-preserving. Still, it is not completely obvious that such deletions are admissible operations, since the latter are defined w.r.t. the structure of STGs, which certainly changes, and since such a deletion can increase the concurrency.

Theorem 4.10 *RedPD is consistency- and auto-cc-preserving, it is an admissible operation and it does not introduce io-conflicts.*

Proof: We have already argued for the first two claims. To check pre-admissibility for the second claim, take an STG S that is without auto-concurrency and dynamic auto-conflicts and satisfies properties (a) and (b) of Definition 4.4.

First assume that the deletion of redundant place p introduces auto-concurrency or an auto-conflict, say the equally labelled transitions t and t' are enabled under the reachable marking M'. Then, $M' = M\big|_{P_i-\{p\}}$ for some reachable marking of S, and t and t' are also enabled under M. Thus, S has auto-concurrency or auto-conflict, a contradiction.

Since the deletion of a place does not add structural conflicts or arcs, it is clear that (a) and (b) of Definition 4.4 are preserved, and the third claim follows from Lemma 4.9. The last claim is also clear.

□

Theorem 4.11 *1. RedTD, i.e. the deletion of an (internal) loop-only or duplicate transition, is consistency- and auto-cc-preserving and it is an admissible operation.*

2. RedTD does not introduce io-conflicts.

Proof: RedTD does not destroy (a) and (b) of Definition 4.4. The reachable markings and the visible transitions they enable before and after the deletion are the same, so the proof is similar to the last. □

We now come to the main operation SecTC, the secure contraction of a transition. To show that this operation is admissible, we give two lemmata corresponding to pre- and full admissibility.

Lemma 4.12 *1. A transition contraction applied to an STG satisfying conditions (a) and (b) of Definition 4.4 preserves these properties.*

2. SecTC is pre-admissible.

3. SecTC applied to an STG satisfying conditions (a) and (b) of Definition 4.4 does not introduce io-conflicts.

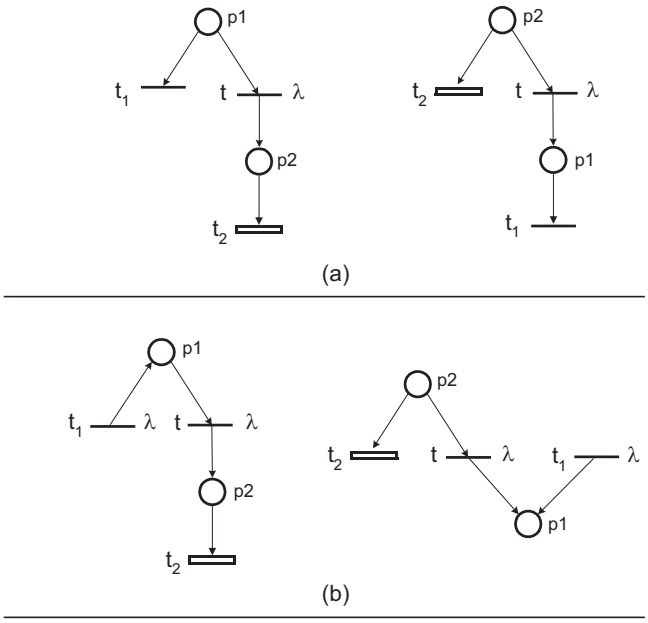


Figure 4

Proof: Although we have changed the definition of a pre-admissible operation a little, this proof is – except for the last part – largely a repetition from [VW02]. The second claim follows from the first and Theorems 3.5 and 3.6. Hence, we concentrate on the first claim now.

So assume contraction of t is applied to some STG S satisfying conditions (a) and (b). Assume the result violates (a) due to some internal transition t_1 and some output transition t_2. Then there are places $pi \in {}^\bullet t_i$ such that $p1 \in {}^\bullet t$ and $p2 \in t^\bullet$ or vice versa; compare Figure 4(a). In the first case, t and t_2 violate (b) in S; in the second case, t and t_2 violate (a) in S.

Finally, assume the result violates (b) due to some internal transition t_1 and some output transition t_2. Then there are places $p1 \in t_1{}^\bullet$ and $p2 \in {}^\bullet t_2$ such that $p1 \in {}^\bullet t$ and $p2 \in t^\bullet$ or vice versa; compare Figure 4(b). In the first case, t and t_2 violate (b) in S; in the second case, t and t_2 violate (a) in S. (Note that in this case the contraction is not secure, but such contractions are considered in the first part, too.)

For the third part, we refer to the third parts of Theorems 3.5 and 3.6. For the latter case observe: if an output transition were involved in a new conflict pair, then we would have a violation of (a) or (b) before the contraction. □

The following is the essential lemma for the treatment of SecTC; it shows what we need (a) and (b) of Definition 4.4 for in our approach. Except for rewriting signals to signal edges, we take the proof from [VW02].

Lemma 4.13 *SecTC applied to some member of a family $(C_i)_{i\in I}$ that satisfies (a) and (b) of Definition 4.4 preserves angelic correctness w.r.t. N.*

Proof: Assume contraction of t is applied to C_j and results in $\overline{C_j}$, $\overline{C}$ resp.; assume further that $\mathcal{B}$ is an angelic bisimulation for N and $(C_i)_{i\in I}$. We define $\overline{\mathcal{B}}$ as $\{(M, \overline{M}) \mid$ there is $(M, M') \in \mathcal{B}$ such that M' and $\overline{M}$ satisfy the marking equality $\}$. Similarly, we will denote by $\overline{M_1}$ the marking of $\overline{C}$ that satisfies the marking equality with the marking M_1 of C.

We check that $\overline{\mathcal{B}}$ is an angelic bisimulation for N and $(C'_i)_{i\in I}$, where C'_j is $\overline{C_j}$ and $C'_i = C_i$ otherwise. Clearly, the initial markings of N and $\overline{C}$ are related since the initial markings of N and C are.

So let $(M, \overline{M}) \in \overline{\mathcal{B}}$ due to $(M, M') \in \mathcal{B}$.

(a) Let $a \in In_N$ and $M[a\pm\rangle M_1$. Either $a \in In_C$ and for some M'_1 $M'[a\pm\rangle M'_1$ and $(M_1, M'_1) \in \mathcal{B}$; then we get $\overline{M}[a\pm\rangle\overline{M_1}$ by Lemma 3.2.3 and $(M_1, \overline{M_1}) \in \overline{\mathcal{B}}$. Or $a \notin In_C$ and for some M'_1 $M'[\rangle M'_1$ and $(M_1, M'_1) \in \mathcal{B}$; again we get $\overline{M}[\rangle\overline{M_1}$ by Lemma 3.2.3 and $(M_1, \overline{M_1}) \in \overline{\mathcal{B}}$.

(b) analogously.

(c) Let $x \in Out_i$ for some $i \in I$ and $\overline{M}\restriction_{P'_i}[x\pm\rangle$. We have two subcases:

$i \neq j$: $\overline{M}$ and M' coincide on $P'_i = P_i$, hence $M'\restriction_{P_i}[x\pm\rangle$.

$i = j$: The image-firing of signal edge $x\pm$ involves an $x\pm$-labelled transition t_1. Since SecTC is pre-admissible, also C'_j satisfies (a) and (b) of Definition 4.4; by (b), only firing internal transitions cannot help to enable t_1, and thus t_1 must already be enabled under $\overline{M}\restriction_{P'_i}$. By (a) and (b) of Definition 4.4 in C_j, no place in ${}^\bullet t_1$ can be in ${}^\bullet t \cup t^\bullet$; therefore, ${}^\bullet t_1$ is the same in C_j and C'_j, $\overline{M}$ and M' coincide on ${}^\bullet t_1$, and $M'\restriction_{P_i}[x\pm\rangle$.

In either case, some M'_1 and M_1 satisfy $M'[x\pm\rangle M'_1$, $M[x\pm\rangle M_1$ and $(M_1, M'_1) \in \mathcal{B}$, and we are done as in (a). □

Together with Theorem 3.4.2, Corollary 3.7 and Lemma 4.5, the above two lemmata give the following result.

Theorem 4.14 *1. SecTC is an admissible operation, and it is consistency- and auto-cc-preserving.*

2. When applied in the algorithm, SecTC does not introduce io-conflicts.

It remains to discuss one vital issue about SecTC: how can we check whether a secure transition contraction has introduced some auto-conflict? There are a number of strategies to deal with this question:

Conservative strategy: In [VW02], it was suggested to restrict attention to STGs without structural auto-conflicts, i.e.: the input specification N has to be free of structural auto-conflicts, and a transition contraction is forbidden if it would create a structural auto-conflict. (Then, instead, backtracking is performed.)

Clearly, we cannot have a dynamic auto-conflict if there is not a structural one. Also, it is easy to check for structural auto-conflicts, since they can only appear in form of new conflict pairs. But just as clearly, this strategy is over-cautious in some cases. In the first place, it makes it impossible to deal with STGs that have some structural auto-conflict initially. In such a case, the specifier is responsible to ensure that there is no dynamic auto-conflict.

Specifier-dependent strategy: Accept inputs with structural auto-conflicts, as long as they are not dynamic ones. A secure transition contraction is only forbidden, if it is of type 2 and a new conflict pair forms a structural auto-conflict. For example, this strategy works well for vmecon, discussed in the next section.

This strategy relies on our new results in Theorems 3.5.5 and 3.6.5. Without these results, the specifier guaranteeing absence of dynamic auto-conflicts would not ensure this absence after some contraction, even if this contraction has not created new conflict pairs. Alternatively, we could try to check that the structural auto-conflict has not turned into a dynamic one due to the contraction, but we would have to repeat this after each contraction – and all this can be avoided relying on Theorems 3.5.5 and 3.6.5. (Recall that only checking new conflict pairs might not be sufficient for general transition contractions.)

Of course, this is still a very conservative strategy, since not each new structural auto-conflict based on a new conflict pair has to be a dynamic one. We would have to check this without constructing the reachability graph e.g. by using place invariants. In many cases the user can possibly find a suitable one easily with human insight into the desired circuit behaviour. Hence, a useful implementation could use the following:

Interactive strategy: Accept inputs with structural auto-conflicts, as long as they are not dynamic ones. Only secure transition contractions of type 2 can be problematic: if such a contraction creates a new conflict pair forming a structural auto-conflict, ask for human intervention; if the user can ensure that there is no dynamic auto-conflict, the contraction can be performed, otherwise it is forbidden – and backtracking is performed instead.

We can add a final touch to our approach. One might worry that human intervention is error-prone: the specifier's guarantee might be wrong or the interactively supplied assertion that a new conflict pair does not give rise to a dynamic auto-conflict might be erroneous; as a consequence, the result of the algorithm might be wrong without anyone noticing. Using the concept of auto-cc-preservation, we have provided results to avoid this problem: Theorems 3.4.2, 4.11 and 4.10 show that a dynamic auto-conflict arising during a run will survive in the form of an auto-conflict or auto-concurrency to the respective final component. When synthesizing a circuit from

this component, one will generate its reachability graph, and during this generation the problem will show up.

As an extreme application of these results, we can run the algorithm simply hoping that new conflict pairs are no problem.

Risky strategy: Accept inputs with structural auto-conflicts, as long as they are not dynamic ones. All secure transition contractions are allowed. In the end, build the reachability graphs of the final components and check that no reachable marking enables two equally labelled transitions; if the check fails, discard the result.

The risk is clearly that all time spent on the algorithm might be wasted. But if our hopes are justified, we get a result with fairly small reachability graphs comparatively cheap. An example where this works is locked2 discussed in the next section.

5 Examples

We will now demonstrate the new features of our improved algorithm for some realistic examples, discussing two of them in some detail; for even more details see http:// www.eit.uni-kl.de/beister/eng/projects/deco_examples/main_examples.html

The decomposition algorithm with the new features presented here has been implemented in our tool DESI[1]; DESI can follow the conservative, the specifier-dependent and the risky strategy. (At the time of writing, the tool only checks for loop-only and duplicate places, but not for redundant places in general.) The table below gives a numbered list of some examples we treated, where NEI is the NEI-arbiter from e.g. [Wol97], while the others are from the benchmark examples. For each example, we give the number of reachable markings and the numbers for the components of the best decomposition we have found so far using the specifier-dependent strategy. (For locked2 we also show a better result obtained with the risky strategy.) Recall that it might already pay off if each of the latter numbers is smaller than the first one. E.g. for tsend_csm, synthesizing a circuit with petrify took 8.12 sec on an Intel Xeon 2.2Ghz with 1 GB memory, while DESI plus petrify on the components took 1.33 sec.

no	name	reach. markings	for the components		
1	locked2	168	70	6	6
	(risky)		32	6	6
2	tsend_csm	36	25	16	
3	mux2	101	50	43	
4	stg_blunno	1241	102	102	44
5	vmecon	24	19	8	
6	pe-send-ifc	117	68	32	
7	NEI	42	21	12	

Examples 1–4 have dummy transitions, which in these examples can now be handled by DESI without any initial human adaption. As we will see below, the two small

[1] http://www.eit.uni-kl.de/beister/eng/projects/download.html

components for locked2 are significantly smaller due to the new operation RedTD; also for mux2, RedTD gives a small improvement. An additional improvement for the large component of locked2 can be obtained with the risky strategy as shown in the second line. Examples 4–8 all have structural auto-conflicts; treatment of these with the decomposition method is only possible due to our improvements. Additionally, stg_blunno also has an io-conflict. NEI cannot be synthesized by petrify, but DESI splits it into a standard ME-element, which was also used in [VW02], and a smaller synthesizable STG.

Let us now have a look at the example locked2 shown in Figure 5. After contraction of the dummy transitions we get Figure 6, which results in Figure 7 as initial component responsible for output *reqa*.

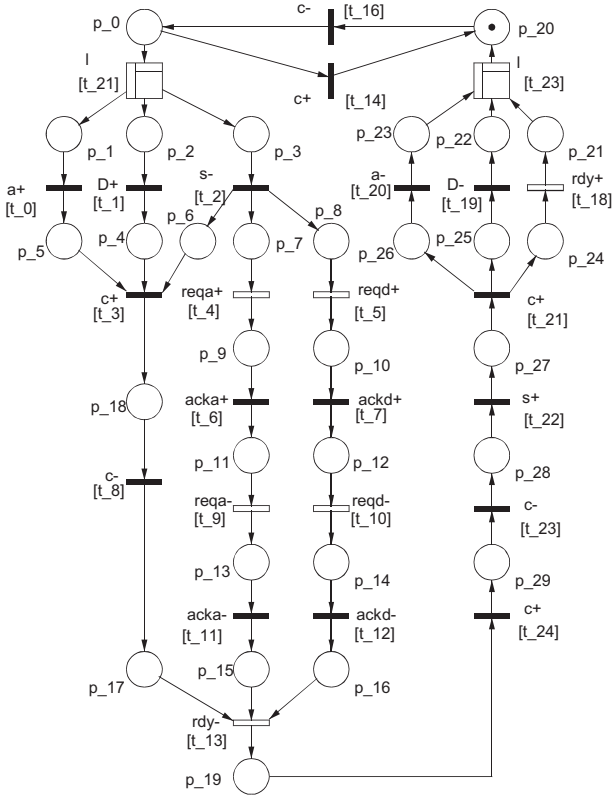


Figure 5

Observe that it is very important that we can get rid of signal c: if we cannot, then contraction of t_0 or t_1 will give a structural auto-conflict between t_3 and t_{14}, such that with a conservative strategy we can also not get rid of signals a and D – giving a comparatively large component. On top of this, we cannot contract t_{14} currently because this would not be a secure contraction.

If we contract the crossed out transitions in Figure 7, we get Figure 8. Here, the marked places are duplicates of each other; hence, we can delete two of them and then contract again the marked transitions – giving Figure 9. The important point is that t_{14} has now turned into a loop-only transition and can be removed. The rest runs smoothly arriving at Figure 10.

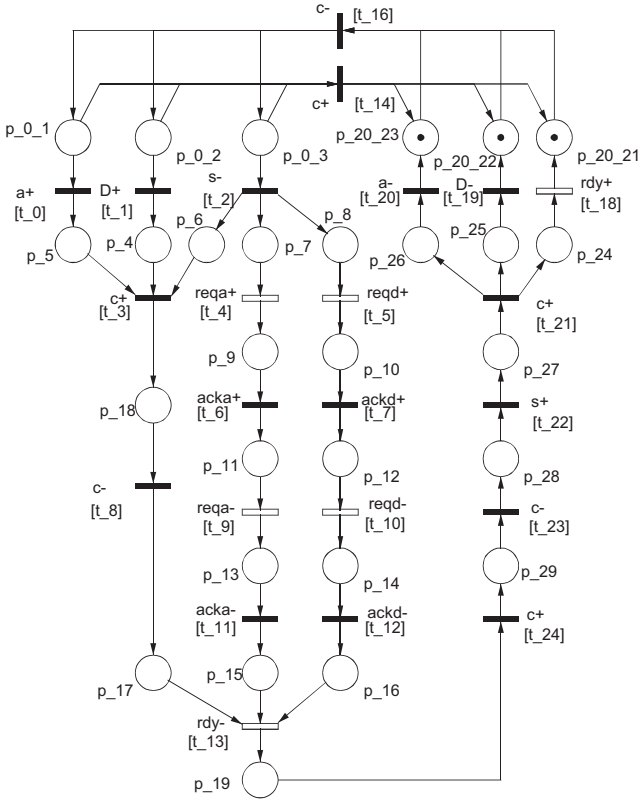


Figure 6

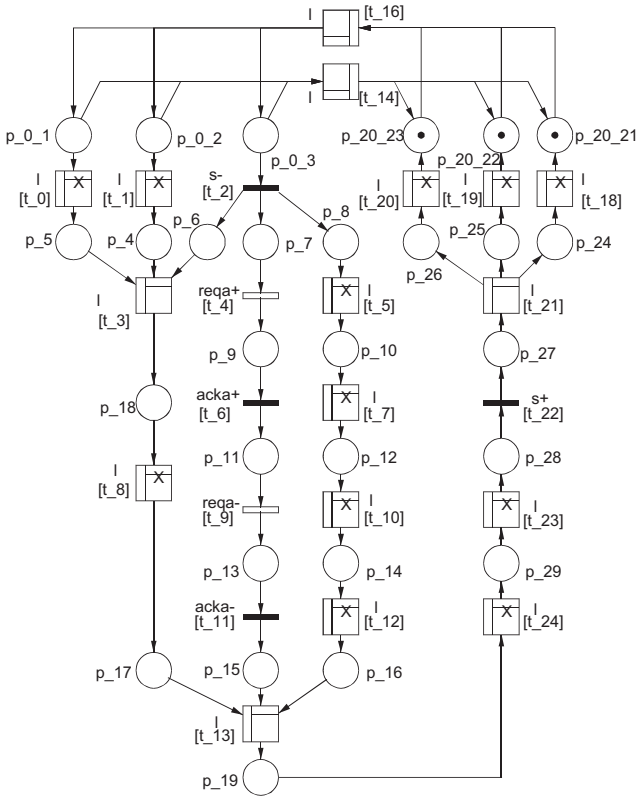


Figure 7

We close this section with a small example where there is some structural auto-conflict initially: vmecon as shown in Figure 11 has a conflict between t_3 and t_{10}. A component producing signal *ds* must see signals *ldtack, drs* and *dsw*, while a component producing signal *lds* must see signals *ds, ldtack, drs* and *dsw*. Hence, the only feasible partition of interest contains ($\{$*ldtack, drs, dsw*$\}$, $\{$*ds, lds*$\}$) and ($\{$*ds, dsw*$\}$, $\{$*dtack*$\}$). For the first component, one has to contract t_0, t_7 and t_9. Since no new conflict pair turns up, the specifier-dependent strategy works without any backtracking – although there always is a structural auto-conflict. For the second component, one finds new conflict pairs when contracting t_3, t_5, t_{10} and t_{11}. But these either involve at least one internal transition or t_4 and t_8; the latter two have the same signal, but different labels, so for the second component the specifier-dependent strategy works without any backtracking as well.

6 Conclusion and Future Work

STG decomposition can help in the synthesis of circuits: it may avoid state explosion, it supports the use of library elements, and it leads to a modular implementation that can be more efficient. [VW02] presented a decomposition algorithm that is much more generally applicable than those known before, and we have presented further improvements:

- We have given a number of results that allow us to deal with structural auto-conflicts: the decomposition algorithm remains correct when there are no dynamic auto-conflicts. Hence, we can carry on even if there are structural auto-conflicts initially or at intermediate stages, provided these are not dynamic ones. If we do so, we only have to check new structural auto-conflicts that turn up in a contraction. If such a check is passed by mistake – e.g. made by the user in an interactive strategy –, this mistake was proven to be recognizable during synthesis.

- We have added the deletion of loop-only transitions to the list of operations allowed in the algorithm.

- We have restructured the correctness proof of [VW02] to cover the previous two items. Already this proof was meant to be extendible to new operations; this did not work out, but it certainly made our restructuring easier. We hope that the new proof structure will turn out to support possible further operations.

- We have extended the algorithm and its correctness proof such that io-conflicts can be dealt with. We have shown that the algorithm does not introduce new io-conflicts, which is important since dynamic io-conflicts are hard to implement.

- We have shown that the algorithm preserves consistency.

- We have noted that we can often deal with dummy transitions in the initial specification, simply by applying transition contraction.

The improvements have been integrated in our tool DESI, and we have demonstrated the usefulness of our improvements with some practical examples.

There are a number of open problems we will tackle in the near future. We will improve the recognition of redundant places in our tool DESI. Often, essentially the same operations have to be performed in different components; DESI should be enabled to reuse the results of some operations. Also, backtracking makes DESI start with a new initial component, while reuse of some operations performed before the backtracking should be possible.

Depending on the choice of a feasible start partition and on the choices of signals for backtracking, we can arrive at different decompositions. We want to study quality criteria (like the overall or the maximal number of reachable markings) for decompositions, and methods to find good decompositions.

We have assumed that the given STG-specification N, its components C_i we have constructed, and their parallel composition C are deterministic. One would get much more freedom in finding components, if one allowed internal signals for the communication between components; then the composition C would not be deterministic anymore, and to treat such STGs it becomes essential to use some form of bisimulation that is not angelic. Also for N, it can be useful to allow nondeterminism: e.g. to treat arbitration in general, it can be necessary to have several enabled and conflicting transitions with the same label, and in [YKK+96], it is argued that auto-conflicts are very useful when modelling so-called OR-causality.

Finally, as argued in [WB00], STGs are not always sufficient to specify the desired behaviour; as an improvement, gSTGs are suggested. Therefore, we want to generalize our approach to gSTGs, and we also want to generalize our correctness criterion to take concurrency into consideration, where it seems to be natural to require the modular implementation to exhibit at least the concurrency prescribed in the specification.

References

[And83] C. André. Structural transformations giving B-equivalent PT-nets. In Pagnoni and Rozenberg, editors, *Applications and Theory of Petri Nets*, Informatik-Fachber. 66, 14–28. Springer, 1983.

[Ber87] G. Berthelot. Transformations and decompositions of nets. In W. Brauer et al., editors, *Petri Nets: Central Models and Their Properties*, Lect. Notes Comp. Sci. 254, 359–376. Springer, 1987.

[BEW00] J. Beister, G. Eckstein, and R. Wollowski. Cascade: a tool kernel supporting a comprehensive design method for asynchronous controllers. In M. Nielsen, editor, *Applications and Theory of Petri Nets 2000*, Lect. Notes Comp. Sci. 1825, 445–454. Springer, 2000.

[Chu86] T.-A. Chu. On the models for designing VLSI asynchronous digital systems. *Integration: the VLSI Journal*, 4:99–113, 1986.

[Chu87a] T.-A. Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications*. PhD thesis, MIT, 1987.

[Chu87b] T.-A. Chu. Synthesis of self-timed VLSI circuits from graph-theoretic specifications. In *IEEE Int. Conf. Computer Design ICCD '87*, pages 220–223, 1987.

[CKK+97] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Trans. Information and Systems*, E80-D, 3:315–325, 1997.

[Ebe92] J. Ebergen. Arbiters: an exercise in specifying and decomposing asynchronously communicating components. *Sci. of Computer Programming*, 18:223–245, 1992.

[KKT93] A. Kondratyev, M. Kishinevsky, and A. Taubin. Synthesis method in self-timed design. Decompositional approach. In *IEEE Int. Conf. VLSI and CAD*, pages 324–327, 1993.

[Lyn96] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, San Francisco, 1996.

[Pet81] J.L. Peterson. *Petri Net Theory*. Prentice-Hall, 1981.

[Rei85] W. Reisig. *Petri Nets*. EATCS Monographs on Theoretical Computer Science 4. Springer, 1985.

[RY85] L. Rosenblum and A. Yakovlev. Signal graphs: from self-timed to timed ones. In *Proc. Int. Work. Timed Petri Nets*, Torino, Italy, 1985.

[Seg93] R. Segala. Quiescence, fairness, testing, and the notion of implementation. In E. Best, editor, *CONCUR 93*, Lect. Notes Comp. Sci. 715, 324–338. Springer, 1993.

[VW02] W. Vogler and R. Wollowski. Decomposition in asynchronous circuit design. In J. Cortadella et al., editors, *Concurrency and Hardware Design*, Lect. Notes Comp. Sci. 2549, 152 – 190. Springer, 2002.

[VYCLdM94] C. Vanbekbergen, C. Ykman-Couvreur, B. Lin, and H. de Man. A generalized signal transition graph model for specification of complex interfaces. In *European Design and Test Conf.*, pages 378–384. IEEE, 1994.

[WB00] R. Wollowski and J. Beister. Comprehensive causal specification of asynchronous controller and arbiter behaviour. In A. Yakovlev, L. Gomes, and L. Lavagno, editors, *Hardware Design and Petri Nets*, pages 3–32. Kluwer Academic Publishers, 2000.

[Wen77] S. Wendt. Using Petri nets in the design process for interacting asynchronous sequential circuits. In *Proc. IFAC-Symp. on Discrete Systems, Vol.2*, Dresden, 130–138. 1977.

[Wol97] R. Wollowski. *Entwurfsorientierte Petrinetz-Modellierung des Schnittstellen-Sollverhaltens asynchroner Schaltwerksverbünde.* PhD thesis, Uni. Kaiserslautern, FB Elektrotechnik, 1997.

[YKK+96] A. Yakovlev, M. Kishinevsky, A. Kondratyev, L. Lavagno, and M. Pietkiewicz-Koutny. On the models for asynchronous circuit behaviour with or causality. *Formal Methods in System Design*, 9:189–233, 1996.

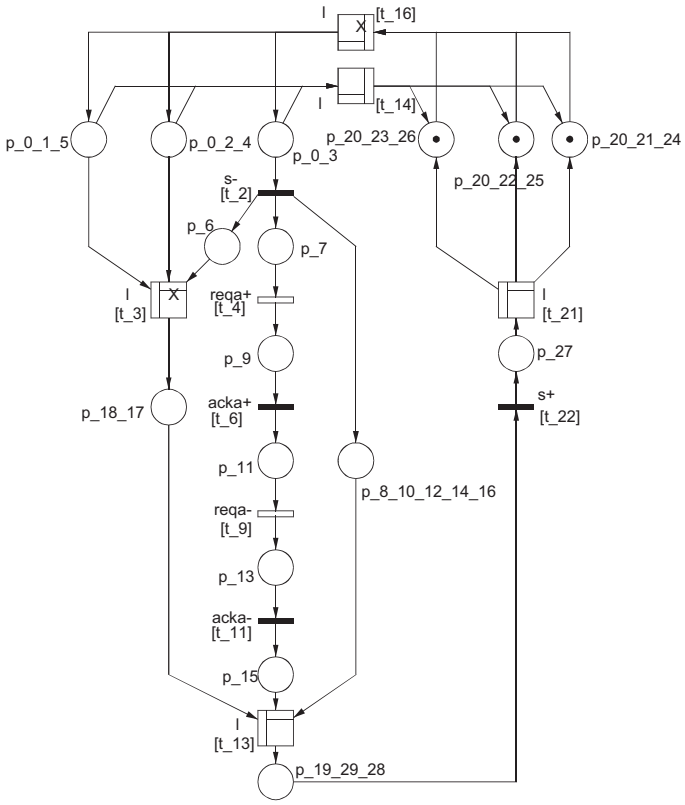

l
[t_16]
l
[t_14]
p_0_1_5
p_0_2_4
p_0_3
p_20_23_26
p_20_22_25
p_20_21_24
s-
[t_2]
p_6
p_7
l
[t_3]
reqa+
[t_4]
l
[t_21]
p_9
p_27
p_18_17
acka+
[t_6]
s+
[t_22]
p_11
p_8_10_12_14_16
reqa-
[t_9]
p_13
acka-
[t_11]
p_15
l
[t_13]
p_19_29_28

Figure 8

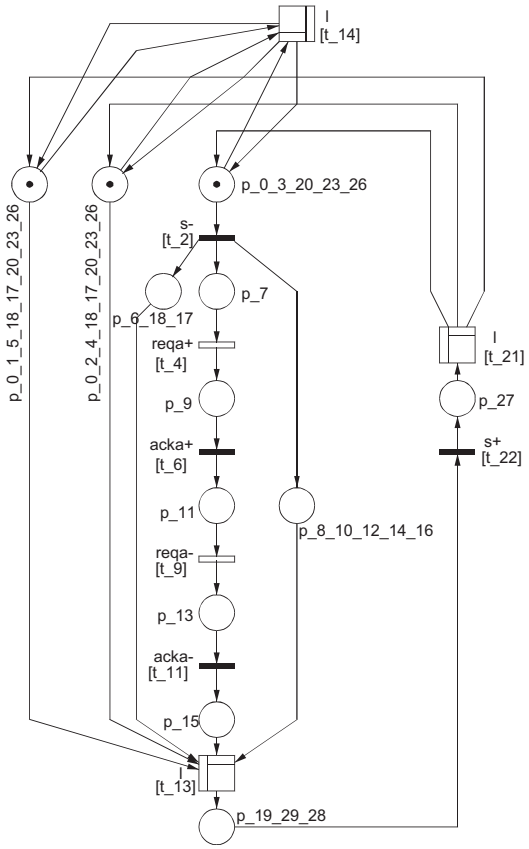


Figure 9

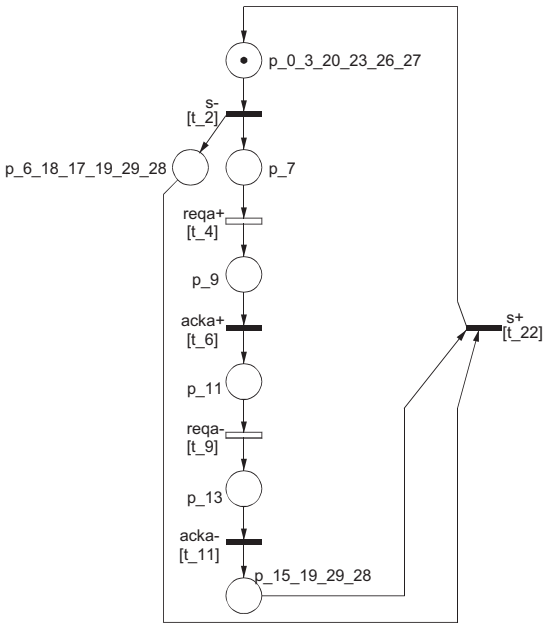


Figure 10

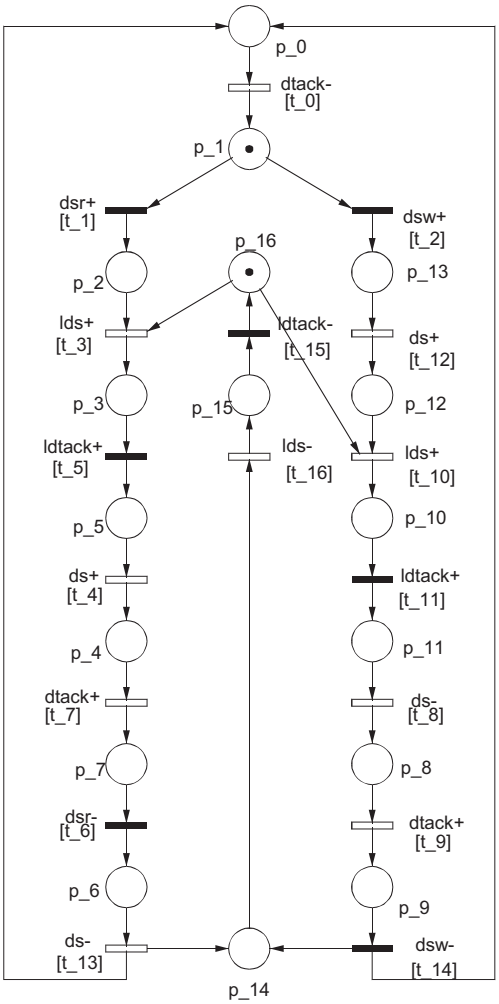

p_0
dtack-
[t_0]
p_1
dsr+
[t_1]
dsw+
[t_2]
p_16
p_2
p_13
lds+
[t_3]
ldtack-
[t_15]
ds+
[t_12]
p_3
p_15
p_12
ldtack+
[t_5]
lds-
[t_16]
lds+
[t_10]
p_5
p_10
ds+
[t_4]
ldtack+
[t_11]
p_4
p_11
dtack+
[t_7]
ds-
[t_8]
p_7
p_8
dsr-
[t_6]
dtack+
[t_9]
p_6
p_9
ds-
[t_13]
dsw-
[t_14]
p_14

Figure 11